Stołeczny Ośrodek
Elektronicznej
Techniki Obliczeniowej

INFORMATYKA mikrokomputerowa

Poznajemy FORTH

Jan Ruszczyc

Warszawa 1987

Książka ta jest przeznaczona dla każdego, kto pragnie poznać ciekawy, wysoce efektywny i bardzo popularny w świecie język programowania FORTH.

Adresowana jest do początkujących miłośników programowania, jak i do profesjonalistów.

W oparciu o liczne przykłady przedstawia metody samodzielnego tworzenia programów. Zawiera obszerny leksykon słów kluczowych czyli instrukcji języka.

FORTH zdobył sobie szerokie uznanie zwłaszcza w programowaniu na mikrokomputerach. Jest wysoce pamięciooszczędny, wielokrotnie szybszy niż BASIC, dialogowy, dowolnie rozszerzalny, dostępny praktycznie na każdy komputer. Zaletom tym nie dorównuje jego znajomość w Polsce.

Wypełnieniu tej luki służyć ma niniejsza książka, pierwszy popularny podręcznik FORTH-a po polsku.

Poznajemy FORTH

Jan Ruszczyc

Druk ZRiWDB Warszawa ul. Królewska 27
Format A-5 Nakład 3000 egz. Ark.13.875
Zam. Nr 230 z 19S7 VIII 12. K-30

P R Z E D M O W A

FORTH jest językiem programowania, zajmującym szczególną pozycję wśród języków wysokiego poziomu. Ma wiele cech, które korzystnie odróżniają go od pozostałych.

Jest kilka do kilkunastu razy szybszy niż, na przykład, BASIC. Tworzy zwarty kod maszynowy gotowy do natychmiastowego wykonania bez potrzeby używania odrębnego kompilatora. jest wysoce pamięciooszczędny. Jest językiem dialogowym, ułatwiającym kontakt użytkownika z komputerem i pod tym względem nie ustępuje BASIC-owi.

Wszystkie te cechy czynią FORTH szczególnie przydatnym w zastosowaniu do mikrokomputerów oraz jednakowo atrakcyjnym dla programistów-amatorów, jak i profesjonalistów.

Zaletą FORTH-a jest jego praktycznie nieograniczona rozszerzalność. Programujący może zrealizować zadania z najrozmaitszych dziedzin - od gier po skomplikowane programy użytkowe. FORTH narzuca metody programowania strukturalnego. Z tego względu ma duże walory edukacyjne i poznawcze.

Ten, kto poznał FORTH, łatwo zrozumie właściwości pozostałych języków algorytmicznych. FORTH, .dostarcza bowiem swoistej filozofii programowania, pomaga w uzmysłowieniu, że jest ono sztuką, zajęciem twórczym, że może być fascynujące. Nie przypadkowo autorzy książek o FORTH określają go dość jednomyślnie tym właśnie przymiotnikiem: fascynujący.

W ostatnich latach popularność FORTH-a w świecie wyraźnie urasta. Przejawem tego jest rozszerzanie się zakresu jego zastosowań, jak i wzrost liczby poświęconych mu publikacji. W Polsce FORTH nie zdobył sobie jeszcze pozycji, na jaką zasługuje. Z przeświadczenia o potrzebie nadrobienia tej luki zrodziła się niniejsza, książka.

Starałem się zrealizować dwa trudne do połączenia cele: umożliwić poznanie FORTH-a osobom, które zetkną się z nim po

raz pierwszy, a jednocześnie dostarczyć informacji o języku pomocnych zaawansowanym programistom. Wynikł z tego podział książki na dwie części. Pierwsza przedstawia FORTH możliwie najprzystępniej, aby umożliwić poznanie języka również osobom, które nie mają, żadnego przygotowania informatycznego. Część druga służy przedstawieniu bardziej złożonych mechanizmów języka.

Książka zawiera liczne przykłady, rysunki a także ćwiczenia pomocne w przyswojeniu jej treści. obszerny rozdział obejmuje przykładowe programy. W aneksie zamieszczono tekst źródłowy kompletnego edytora, a także obszerny leksykon słów kluczowych FORTH-a oparty na aktualnych standardach i uwzględniający różnice w nazewnictwie.

Wykorzystano najnowsze dostępne publikacje o FORTH, wymienione w bibliografii. Przy powoływaniu się na prace, liczby w nawiasach kwadratowych wskazują pozycje w bibliografii. Przykłady i ćwiczenia zostały przygotowane i sprawdzone na Atari 8OOXL z pomocą udanej implementacji wykonanej w oparciu o standard fig-FORTH-a przez Patricka L. Mullarky,ego i nazwanej Extended fig-FORTH.

Ze względu na daleko posuniętą standaryzację języka nie ogranicza to ogólnego charakteru książki. FORTH w niej przedstawiony - to FORTH na wszystkie komputery.

W warunkach znanej niejednolitości polskiej terminologii informatycznej kierowałem się dążeniem, by wszędzie gdzie jest to możliwe, stosować określenie bliskie rodzimemu słownictwu. Na przykład zamiast „pozycyjny system numeryczny" użyto „pozycyjny układ liczbowy", zamiast angielskiego „interpretera” - pochodzące z łaciny, a dobrze znane z pokrewnych kontekstów słowo „interpretator".

Wdzięczny będę za uwagi Czytelników o treści książki, kierowane do Wydawcy lub bezpośrednio do mnie.

Jan Ruszczyc

Adres: ul. Wilcza 25 m. 6,
00-544 Warszawa
tel. 21-79-05

ROZDZIAŁ 1
DLACZEGO FORTH?

1.1 Trochę historii

Wynalazcą języka programowania FORTH jest Amerykanin Charles H. Moore. Określenie "wynalazca" jest tu w pełni stosowne, bowiem Moore stworzył zarówno nowatorską koncepcję języka, jak i zrealizował ją w praktyce, dokonał pierwszej implementacji FORTH-a czyli jego wprowadzenia na konkretny komputer.

Był nim IBM 1130, a stało się to pod koniec lat sześćdziesiątych. W szybkim rozwoju informatyki tamte czasy można uznać za zamierzchłe. Mimo to podstawowe założenia nowego języka utrzymały się do dziś w niezmienionej postaci. Rozwój FORTH-a, który trwa nadal, zmierza w kierunku znajdowania coraz to nowych możliwości jego zastosowań, a nie zasadniczej modyfikacji samego języka.

Moore był przekonany, że stworzył język, który lepiej niż jakikolwiek inny pozwala rozwiązywać zadania programistyczne na małych komputerach. Pogląd ten wyraził już w tytule artykułu "FORTH: A New Way to Program MiniComputers" ("Forth nowa droga programowania na minikomputerach") , który był pierwszą szerszą prezentacją nowego języka. "Celem FORTH-a - pisał Moore - jest zmaksymalizowanie korzyści, jakie daje programistom, i zminimalizowanie kosztów w sensie nakładów czasu i pamięci".

Wspomniany artykuł ukazał się w roku 1974 w suplemencie pisma "Astronomy and Astrophysics" [1] . Nie jest to przypadkowe. Pierwszym szerszym zastosowaniem praktycznym FORTH-a stało się wykorzystanie go w roku 1971 do przetwarzania danych w Krajowym Obserwatorium Radioastronomicznym USA. W pięć lat później komitet Międzynarodowej Unii Astronomicznej uznał FORTH za standardowy język programowania w astronomii.

Jak powstała nazwa FORTH? Często podaje się następujące wyjaśnienie. Moore miał być przekonany, że wynalazł język czwartej (fourth) generacji. Ponieważ jednak IBM 1130 akceptował tylko nazwy pięcioliterowe, obrał skróconą wersję liczebnika. Istnieją również skromniejsze wyjaśnienia. Glyn Emery[4] pisze, iż Moore nazwał tak język ze względu na to, że zadowalający wynik osiągnął w czwartej próbie, a posłużył się do wykonania pierwszego interpretatora językiem FORTRAN. Ponoć dlatego również nazwę FORTH, przez nawiązanie do FORTRANU, wymawia się z twardą końcówką, jak polskie słowo "fort". Nawiasem mówiąc FORTH jest całkowicie inny niż FORTRAN, a trzon kompilatorów tworzy się dziś przy użyciu języka maszynowego.
Tak czy owak FORTH stał się istotnie rewelacją, a krąg jego zwolenników szybko się rozszerzał. Sam Moore założył instytucję wytwarzającą programy o nazwie FORTH Inc. Doszło też do powstania FORTH Interest Group w USA, która odgrywa obecnie dużą rolę w upowszechnianiu języka i krzewieniu wiedzy o nim, opracowuje programy, wydaje pisma i książki specjalistyczne, w tym podręczniki instrukcyjne zawierające szczegółowe opisy języka i jego oprogramowania użytkowego. Przedrostek "fig" zaznacza, że implementacja dokonana jest w oparciu o zalecenia FORTH Interest Group.
FORTH nie ma oficjalnego standardu światowego, jednak do jego jednolitości przyczynia się fakt, że podstawowe rozwiązania fig-FORTH-a akceptuje kalifornijski FORTH Standard Team ustalający kolejne standardy, ostatni w roku 1983. Dzięki temu implementacje FORTH-a dokonywane na dziesiątki komputerów w różnych krajach mają taki sam podstawowy zasób słów, a posiadacze rozmaitych komputerów stosunkowo łatwo mogą ustalić wspólny język.
Reasumując, FORTH wyniósł z dotychczasowych lat swego rozwoju cenną zaletę: jest językiem ustabilizowanym, dziś już w pełni dojrzałym, a jednocześnie prężnym i dynamicznym.

1.2 Jaki jest FORTH?

Niniejsza książka ma na celu zapoznanie Czytelnika z FORTH-em. Na wstępie przedstawmy zwięźle jego wizytówkę.

FORTH należy do języków programowania określanych mianem języków algorytmicznych wysokiego poziomu. Podobnie jak pozostałe spośród nich stanowi narzędzie, które ułatwia i usprawnia programowanie. Zastosowane w FORTH oryginalne rozwiązania nadają mu nieco odrębną pozycję. Główne swoiste cechy FORTH-a to:

- oparcie się na podstawowej jednostce programowej zwanej słowem, któremu w innych językach odpowiadają pojęcia komendy, podprogramu, a także stałej lub zmiennej;
- skupienie zarówno słów predefiniowanych, jak i tworzonych przez użytkownika w jednej wspólnej strukturze, dającej się dowolnie rozszerzać, aż po granice pamięci komputera, zwanej słownikiem;
- kompilowanie tworzonych słów w postaci kodu gotowego do natychmiastowego wykonania;
- posługiwanie się we wszystkich obliczeniach pośrednią strukturą stosu;
- stosowanie w podstawowej wersji dwubajtowych liczb całkowitych ze znakiem z możliwością korzystania również z liczb bez znaku, liczb podwójnych (czterobajtowych) , a także arytmetyki zmiennopozycyjnej;
- stosowanie odwrotnej notacji polskiej (ang. Reverse polish notatioin) (znaki działań po liczbach).

Funkcjonowanie wszystkich tych specyficznych mechanizmów języka będzie omówione w dalszych rozdziałach.

Jak przebiega programowanie w FORTH? Polega ono na tworzeniu nowych słów z pomocą słów wcześniej zdefiniowanych. Proces ten trwa do chwili, gdy ostatnie słowo obejmie i scali poprzednio stworzone. Napisanie tego głowa i naciśnięcie klawisza Return (Enter na Spectrum) wywoła program.

Programowanie w FORTH przypomina zatem wkładanie mniejszych pudełek do większych w najdogodniejszych zestawach, aż wreszcie ostatnie, największe pudełko staje się programem. W informatyce określamy to jako programowanie od dołu do góry.

FORTH nie przyjmuje nie zdefiniowanego słowa, czyli procedury, zmiennej lub stałej. Zaleca się, by poszczególne słowa nie były zbyt długie. Odpowiada to zasadom programowania strukturalnego, ułatwia zrozumienie budowy programu, a także jego uruchamianie i testowanie.

1.3 Podstawowe cechy

Czytelnik, którego zachęca się do poznania FORTH-a, może nie bez racji zapytać: po co mam się uczyć innego języka programowania? co mi to da? czy nie wystarczy mi, na przykład, znajdujący się w moim komputerze BASIC, którego w dodatku jeszcze w pełni nie poznałem? Postarajmy się możliwie konkretnie i przekonująco odpowiedzieć na te wątpliwości. Odpowiedź najogólniejsza, a przez to, być może, nie w pełni trafiająca do przekonania, brzmiałaby następująco: w warunkach współczesnego burzliwego rozwoju informatyki i przenikania jej do wszystkich dziedzin życia, w tym również do naszych ognisk domowych, poznawanie języków programowania staje się ważnym składnikiem podnoszenia ogólnego poziomu wiedzy każdego Polaka, w tym zwłaszcza wchodzących w życie pokoleń. Poznawanie języków programowania rozszerza horyzonty i pozwala lepiej rozumieć świat, zwiększa podobnie jak poznawanie języków obcych - możliwości udziału w postępie cywilizacyjnym dokonującym się wokół nas. Człowiek ograniczający się do znajomości jednego języka programowania (a spotykamy takich, niestety, choć coraz rzadziej, również wśród zawodowych programistów), ogranicza tym samym własne szanse i pozostaje w tyle. Ponieważ te argumenty nie do każdego przemówią, użyjmy innych. Wskażmy na kilka podstawowych cech FORTH-a, które decydują o jego wysokiej przydatności i o tym, że pod niektórymi względami góruje nad innymi językami programowania.

1.3.1 Szybkość

Wykonajmy prosty test, by przekonać się, jak szybko liczy BASIC, a jak FORTH.

Oto miniprogram W BASIC-u:

```
10 FOR I = 1 TO 30000 : NEXT I : BEEP
```

Jeżeli nie mamy wbudowanego brzęczyka, musimy patrzeć, kiedy na ekranie pojawi się napis READY. Uruchommy program i zanotujmy czas wykonania.

A teraz to samo w FORTH.

```
: TEST 30000 0 DO LOOP BEEP ;
```

Zdefiniowaliśmy słowo TEST o takim samym działaniu, jak jednowierszowy program w BASIC-u. Treść definicji i sposób jej tworzenia omówimy później. Napiszmy teraz TEST i od naciśnięcia klawisza RETURN mierzmy czas do sygnału dźwiękowego lub pojawienia się napisu "ok".

Tak, to nie pomyłka. Na każdym komputerze FORTH wykonuje ten program wielokrotnie, 8-15 razy szybciej niż BASIC. na ATARI 800XL BASIC liczył przez 58 sekund, a FORTH - 3 sekundy czyli przeszło 19 razy szybciej.

Ogólnie biorąc, im większej ilości obliczeń wymaga program, tym bardziej FORTH góruje szybkością nad językiem BASIC, także nad najnowocześniejszymi jego wersjami, takimi np. jak BASIC XL, w których zastosowano istotne usprawnienia.

Ocenia się, że FORTH jest przeciętnie 8-10 razy szybszy niż BASIC. Ponadto FORTH przyspiesza sam proces programowania.

Owen Bishop [2] ocenia, że w porównaniu z innymi językami wysokiego poziomu te same zadania można zaprogramować w FORTH dwukrotnie szybciej, natomiast w asemblerze FORTH-a nawet dziesięciokrotnie szybciej niż w standardowych asemblerach.

Szybkość działania FORTH-a czyni z niego język wręcz idealny do programowania gier komputerowych i nic dziwnego, że właśnie w tej dziedzinie znalazł szerokie profesjonalne zastosowania. Duże obliczenia, które BASIC "męczy" przez dziesiątki minut, można wykonać z pomocą FORTH-a w kilkadziesiąt sekund. Podobne przykłady znajdziemy również w tej książce.

1.3.2 Półkompilacja

Duża szybkość wykonywania się programów w FORTH wynika przede wszystkim z dwóch przyczyn. Jedną jest posługiwanie się stosem i arytmetyką stałopozycyjną. Drugą i zasadniczą jest to, że nowo definiowane słowa FORTH-a są natychmiast poddawane kompilacji i w tej postaci, jako gotowe do niezwłocznego wykonanie, wprowadzane do słownika.
By to bliżej wyjaśnić, wskażmy, że wśród języków wysokiego poziomu występują na ogół dwa typy: języki interpretowane i kompilowane. Na przykład, BASIC jest językiem interpretowanym. Za każdym razem, gdy uruchamiamy program, interpretator BASIC-u. od nowa rozpoczyna tłumaczenie tego programu na język właściwy maszynie cyfrowej. Komputer wykonuje instrukcje w kolejnych liniach, przeskakuje do innych linii, gdy napotka GOTO, sprawdza warunki, realizuje pętle itd. Wszystko to wydłuża czas wykonywania programu. Ma jednak również zaletę. W języku interpretowanym możemy zatrzymać program, dokonać niewielkiej poprawki i uruchomić go znowu. Inaczej działa język kompilowany, na przykład, FORTRAN. Kiedy napiszemy już cały program, nie możemy go natychmiast uruchomić, lecz musimy z pomocą kompilatora przekształcić ten program, zwany źródłowym, w kod maszynowy. Gdy potem uruchomimy taki kod, działa on istotnie szybko. Natomiast chcąc wprowadzić drobną poprawkę musimy dokonać jej w programie źródłowym, a potem program skompilować od nowa. FORTRAN ze względu na szereg zalet ceniony jest przez zawodowych programistów. Natomiast w amatorskich zastosowaniach jest nieco kłopotliwy, A jak to jest rozwiązane w FORTH? Nie jest on w tradycyjnym sensie ani językiem interpretowanym, ani kompilowanym. Interpretator FORTH-a zdolny jest wykonywać o b i e funkcje. Zależy to jedynie od fazy, w jakiej znajduje się system. Gdy definiujemy nowe słowo, FORTH jest w fazie kompilacji, to znaczy buduje kod, który po zakończeniu definiowania słowa staje się częścią, słownika. Nie jest to kod maszynowy taki sam, jak ten, który wytwarzają zwykłe asemblery, lecz noszący trudną do przetłumaczenia angielską nazwę:

threaded code. Można to określić jako kod zszywany lub nizany. Odkładając na później wyjaśnienie sprawy stwierdźmy, że w związku z tą szczególną formą kodu czynność wykonywana przez FORTH nazywana jest często półkompilacją. FORTH w tworzeniu nowych słów posługuje się słowami uprzednio zdefiniowanymi. Około 60 z nich - to słowa pierwotne (primitives) napisane w języku maszynowym. Wszystkie następne, w tym również w większości słowa mające charakter komend języka, definiowane są w FORTH. Jest on pod tym względem językiem niezwykłym, bowiem tworzy sam siebie. Niektórzy żartobliwie pytają, czy w ogóle FORTH istnieje. Jest to, oczywiście, żart, ale spróbujmy w pamięci komputera znaleźć interpretator FORTH-a wykonujący tak wiele złożonych zadań. Nie jest to łatwe, bowiem doszukamy się co najwyżej... niespełna 30 bajtów. Dodajmy jednak, że bardzo wiele pracy wykonują niektóre predefiniowane słowa FORTH-a. Może z nich skorzystać użytkownik, lecz spełniają również dużą rolę w funkcjonowaniu systemu. Praktycznie kryją w sobie interpretator.

1-3.3 Pamięcioooszczędność

Zdolność do tworzenia wyjątkowo zwięzłego kodu, zwarta budowa słownika, wykonywanie obliczeń za pośrednictwem stosu, z którego liczby wykorzystane natychmiast znikają, możliwość wielokrotnego posługiwania się tymi samymi słowami i inne cechy FORTH-a sprawiają, że jest on językiem wysoce pamięcioooszczędnym i dzięki temu szczególnie przydatnym na mikrokomputery. Alan Winfield trawestując tytuł artykułu Ch. H. Moore'a opatrzył swą książkę podtytułem: "A New Way to Program Microcomputers" ("Nowa droga programowania na mikrokomputerach") [6]. Dobrze oddaje to fakt, że właśnie w okresie szerokiego upowszechniania się komputerów domowych o stosunkowo niedużej pamięci operacyjnej FORTH w szczególnej mierze ujawnia dodatnie strony faktu, iż z pomocą stosunkowo małej liczby komórek pamięci potrafi zdziałać bardzo wiele. Nie dajmy się zwieść faktowi, że powstają teraz mikrokomputery o dłuższym słowie i większej pamięci. Po pierwsze, nadal będziemy u nas szeroko korzystać z komputerów 8-bitowych, bo

nie wyrzucimy ich przecież w pełni sprawnych na złom. Po drugie, pamięć komputera będzie zawsze w cenie, zwiększą się jedynie zadania. FORTH potrafi im podołać.

1.3.4 Rozszerzalność

Jedną z najbardziej niezwykłych cech FORTH-a jest to, że. możemy go dowolnie rozszerzać. Podczas gdy inne języki programowania narzucają użytkownikowi rygorystyczny system komend i metod postępowania, FORTH umożliwia wybór własnych dróg programowania, tworzenie własnych typów danych, definiowanie w obrębie słownika Własnych specjalistycznych podsłowników. Każde zadanie programistyczne, które zechcemy podjąć, może być wykonane w FORTH. Źródła jego wszechstronności i plastyczności nie tkwią w istocie w roli i znaczeniu poszczególnych słów i innych konkretnych rozwiązań, lecz w czymś, co można nazwać filozofią języka. FORTH jest językiem zwróconym twarzą w stronę wszystkich, którzy informatykę traktują jako zajęcie twórcze, którzy chcą eksperymentować, szukać nowych dróg rozwiązywania nurtujących ich problemów, a także... traktować programowanie jako relaks, zabawę. Jest on językiem pobudzającym zmysł nowatorstwa. Z drugiej strony rozszerzalność języka umożliwia tworzenie jego prostszych, łatwiej przyswajalnych, dostępnych dla zupełnego laika wersji i odgałęzień. Początkujący może nadać słowom nazwy pomagające mu się nimi posługiwać. Stworzenie "polskiego FORTH-a" byłoby zadaniem dziecinnie łatwym. Wystarczy napisać

```
: ZMIENNA VARIABLE ;
: TU      HERE     ;
: KASUJ   DROP     ;
: LICZBA  NUMBER   ;
```

i tak dalej, by przypisać polskie nazwy funkcjom ich nazwanych po angielsku pierwowzorów.
Wbrew pokutującym opiniom, że FORTH jest trudny, wielu zwolenników tego języka widzi możliwości jego zastosowania w informatycznej edukacji dzieci i Młodzieży. Jedno jest w

każdym razie niewątpliwe: z pomocą FORTH-a można wpoić zasady porządnego, logicznego i twórczego programowania. Nawiasem mówiąc, niełatwo jest w FORTH doprowadzić do "zatkania się" pamięci dość często zdarzającego się w LOGO, które należy do języków rozrzutnych w korzystaniu z ograniczonej pamięci mikrokomputera.

1.3.5 Dialogowość

Jedną z możliwości, z których chętnie korzysta zwłaszcza programista-amator i początkujący użytkownik komputera, jest wprowadzanie komend, które zostaną natychmiast wykonane, zadawanie komputerowi pytań i otrzymywanie niezwłocznych odpowiedzi. Mówimy o jednych językach, że są dialogowe czy konwersacyjne, o innych natomiast, że takimi nie są. Ogólnoświatowa popularność BASIC-u wśród szeregowych miłośników informatyki wynika przede wszystkim z faktu, że w języku tym łatwo jest "pogadać sobie" z komputerem, że jest to język prosty i dialogowy. Do języków dialogowych należy takie bez wątpienia LOGO. Natomiast sporo języków mających duże znaczenie w zawodowym programowaniu, takich jak FORTRAN, PASCAL czy COBOL, pozbawionych jest waloru dialogowości. FORTH jest w pełnym tego słowa znaczeniu językiem dialogu,

1.3.6 Inne cechy

FORTH pobudza do programowania strukturalnego. Szerzej omówimy tę sprawę w rozdziale poświęconym konstrukcjom. Tu zaznaczmy jedynie, że programowanie strukturalne uznać można za podstawowy warunek tworzenia programów przejrzystych, zrozumiałych i dobrze funkcjonujących. W programowaniu strukturalnym (termin ten nie jest zresztą w pełni jednoznaczny) obok wykonywania instrukcji sekwencyjnie, to znaczy po kolei, jedna za drugą, przewiduje się kilka podstawowych konstrukcji powodujących zmianę sekwencyjnego porządku wykonywania. Zasadą jest tworzenie programu z niedużych fragmentów zwanych

modułami, które mogą być również zagnieżdżane we wnętrzu innych modułów. Nie zaleca się stosowania skoków GOTO i dlatego programowanie strukturalne nazywane jest niekiedy, nie całkiem ściśle, programowaniem bez GOTO.

Otóż w języku FORTH nie można w istocie programować inaczej niż strukturalnie. Zawiera on, podobnie jak PASCAL, wszystkie podstawowe struktury zmiany sekwencyjnego porządku wykonania: rozwidlenia warunkowe i pętle. Nie ma w nim, bo nie jest potrzebna, instrukcji GOTO. Modułowość programu narzuca sama budowa słownika.

Ostatnia wreszcie cecha, o której należy wspomnieć w tym pobieżnym przeglądzie - to p r z e n o ś n o ś ć FORTH-a czyli duża łatwość przenoszenia napisanych w nim programów z jednego komputera na inny. Tylko w przypadku, gdy występują podprogramy napisane w języku asemblera, należy je "przetłumaczyć" na nowy asembler. Mogą ponadto występować nieznaczne, przy pewnym doświadczeniu łatwe do pokonania różnice w zestawie słów predefiniowanych. Poza tym programy napisane na rozmaite komputery, zwłaszcza gdy posługują się jednolitym słownictwem fig-FORTH-a i bardzo do niego zbliżonym standardem FORTH-83, są identyczne.

1.4 Gama zastosowań

Współczesną pozycję języka FORTH charakteryzuje rozległość i różnorodność jego zastosowań.

Już pierwsze zastosowania w radioastronomii potwierdziły efektywność FORTH-a jako tzw. języka czasu rzeczywistego, tzn. zdolnego szybko przetwarzać napływające dane. Zaleta ta sprawiła, że FORTH jest m. in. wykorzystywany w sterowaniu robotami i urządzeniami przemysłowymi.

Szeroko stosuje się FORTH w urządzeniach wytwarzających efekty specjalne w filmach. Sprawdził się on pod tym względem w szczególnie bogatych w efekty filmach z dziedziny sciencefiction, takich jak "Star Wars", "Battle Beyond Stars" i "Star Trek".

Firma Atari rozwinęła w oparciu o FORTH system testów do gier arkadowych, czyli odznaczających się akcją o największej

szybkości. Liczne gry arkadowe dostępne na rynku pracują w FORTH. Atari wytwarza także gry telewizyjne w FORTH oraz oferuje specjalną wersję języka: Game FORTH.
Wielki szpital w Los Angeles, Cedar Sinai Medical Center, używa programów w FORTH na PDP 11/60 do:

- kontroli 32 terminali;
- gromadzenia danych o pacjentach z pomocą czytnika optycznego;
- statystycznego opracowywania tych danych;
- bieżących analiz krwi i kardiologicznych.

Szpital rozwinął przenośny układ opieki nad chorym zaprogramowany w FORTH.
Kieszonkowy tłumacz Craig M100, na którym z pomocą wymiennych modułów można dokonywać przekładów w obrębie około dwudziestu języków, został opracowany w FORTH.
W FORTH działają urządzenia na satelitach amerykańskich służące do badania temperatury i wymiany danych telemetrycznych. Specjalna wersja tego języka znalazła zastosowanie w amatorskim radiosatelicie OSCAR Phase III.
Dialogowy charakter FORTH-a i jego zdolność do pracy w ograniczonej pamięci operacyjnej skłoniły informatyków radzieckich do posłużenia się tym językiem przy tworzeniu systemu redagowania tekstów w kompleksie terminalowym EC797O obejmującym jednostkę centralną i do 32 terminali. W systemie tym translator-interpretator języka FORTH steruje działaniem okien tekstowych służących do tworzenia tekstu wyjściowego, komponowaniem jego kolejnych stron, zarządza bazą danych edytora. Patrz praca [19] . Również w ZSRR FORTH znalazł inne, niejako pośrednie, a niezmiernie ciekawe zastosowanie. Zespół naukowców uniwersytetu moskiewskiego pod kierunkiem M. P. Brusiencowa potraktował filozofię i koncepcję FORTH-a jako punkt wyjścia do stworzenia własnego Dialogowego Systemu Programowania Strukturalnego. System ten służy do budowy strukturalizowanych programów z pomocą mikrokomputerów. Naukowcy radzieccy opracowali dla potrzeb tego systemu własny język programowania, w którym w pełni wykorzystali takie

podstawowe składniki i cechy FORTH-a, jak stos danych, słownik, kod zszywany i in. Równocześnie celem zabiegów stało się połączenie zalet FORTH-a z możliwością programowania strukturalnego "od góry do dołu". Patrz [18] , [19] .
Sumując: w swej wersji podstawowej FORTH potwierdził swą praktyczną przydatność w bardzo różnorodnych dziedzinach. Stał się zarazem dla teoretyków i praktyków inspiracją do dalszych poszukiwań. Wszystko to przemawia dobitnie za tym, by go poznać.

1.5 Implementacje

Popularność FORTH-a i jego szczególna przydatność w pracy na mikrokomputerach i minikomputerach sprawiły, że nie ma dziś w istocie bardziej znaczącego komputera tych klas, na który nie powstałby co najmniej jeden, a nieraz kilka interpretatorów tego języka. Salman, Tisserand i Toulout [5] podają zestawienie, z którego wynika, że FORTH implementowany jest na 8 mikroprocesorów 8-bitowych i siedem 16-bitowych oraz na ok. 40 mikrokomputerów, w tym na wszystkie najpopularniejsze w Polsce, takie jak Atari, Commodore, Spectrum i Apple II. Interpretatory wgrywane są z taśmy lub dyskietki. Istnieją również mikrokomputery, jak brytyjski Jupiter Ace, z wbudowanym interpretatorem FORTH-a.
Informacje pojawiające się w czasopismach wskazują, że w zasadzie każdy nowy komputer wprowadzany do sprzedaży ma w swoim wyposażeniu programowym interpretator FORTH-a. Każdy zatem w Polsce, kto pragnie zainteresować się FORTH-em, może niewielkim nakładem wysiłku i ewentualnie kosztów lub w drodze zamiany wyposażyć się w interpretator - niezbędne narzędzie wyjściowe do poznawania FORTH-a i programowania w tym języku. Poczynić tu jednak trzeba parę uwag. Z tego, co powiedzieliśmy wcześniej, wynika, że budowa "chodzącego" interpretatora FORTH-a jest stosunkowo łatwa. Zachęcam wszystkich amatorów programowania do podjęcia tego zadania. Jednakże interpretator chodzący nie musi być jeszcze dobry. W dodatku - i to jest główna przyczyna kłopotów - rozmaici pożałowania godni "crackerzy" psują interpretatory przez niedokładne i niekompletne ich

kopiowanie. Chroniczną luką jest brak firmowych instrukcji. Jest to szczególnie kłopotliwe wtedy, gdy twórcy interpretatora wprowadzili w nim jakieś pożyteczne innowacje. Któż np. z użytkowników wspomnianego już Extended fig-FORTH będzie wiedział, że znakomita komenda SAVE bez jakichkolwiek dodatkowych czynności wpisze mu cały interpretator i słownik wraz z nowo zdefiniowanymi słowami na dyskietkę? A zacznie to od sektora pierwszego, wprowadzając do niego dane powodujące potem automatyczne wgranie interpretatora po włączeniu komputera. Jeżeli nie zna się tych szczegółów, to przy wgrywaniu z pomocą SAVE można poniszczyć programy wcześniej zapisane na dyskietce. Ogólna rada: gdy nie znamy interpretatora, obchodzimy się z nim ostrożnie. Przy pewnej pomysłowości i wiedzy można z reguły uporać się z początkowymi trudnościami. Książka niniejsza będzie w tym pomocna.

1.6 Oprogramowanie systemowe

Istotną cechą FORTH-a jest to, że bierze niejako w posiadanie cały komputer, tworzy w nim własną specyficzną organizację pamięci, logicznie dzieli ją na wyspecjalizowane pola, organizuje własny mechanizm współpracy z urządzeniami zewnętrznymi: magnetofonem, drukarką, a przede wszystkim stacją dyskietek. Z pomocą szeregu użytecznych słów FORTH czyni z dyskietki łatwo dostępne przedłużenie pamięci komputera, miejsce przechowywania i zapisywania programów.

Podstawowy zasób FORTH-a to minimalny interpretator wraz z predefiniowanym podstawowym słownikiem, a ściślej podsłownikiem, noszącym taką samą jak język nazwę: FORTH.

Jednakże w zasobie FORTH-a mogą i powinny mieścić się również inne ważne instrumenty pomocnicze składające się na oprogramowanie systemowe. Wymieńmy najważniejsze z nich.

Debugger, czyli program uruchamiający - to zazwyczaj nieduży zestaw słów włączanych wprost do podsłownika FORTH, pomagających w tworzeniu programów. Należą do niego m. in. słowa pozwalające ustalić aktualny stan stosu, wartość liczby w układzie szesnastkowym, a wreszcie, co jest niezmiernie

pożyteczne, wewnętrzną budowę poszczególnych procedur. Definicje wielu słów debuggera znajdzie Czytelnik w niniejszej książce.

E d y t o r - to z reguły osobny podsłownik zawierający słowa pomocne w redagowaniu programu i jego zapisywaniu na dyskietkę. Prosty, łatwy do zrealizowania edytor przedstawiamy w aneksie.

A s e m b l e r - to wewnętrzny asembler FORTH-a, z którego pomocą możemy w obrębie języka, bez konieczności stosowania innych środków, budować wstawki maszynowe w postaci takich samych słów jak inne słowa FORTH-a. Jak wiemy, nagminną praktyką w BASIC-u jest włączanie kodu maszynowego w postaci DATA. W FORTH nie jest to potrzebne.

Dostępne są w FORTH pakiety służące do stosowania arytmetyki zmiennopozycyjnej, obsługi kanałów systemu operacyjnego komputera, obsługi dyskietek z ich formatowaniem i kopiowaniem włącznie.

Istotne znaczenie mają zestawy procedur służących do operowania grafiką i dźwiękiem. Każdy niemal komputer ma w tej dziedzinie inny system sterowania. By zatem w pełni wykorzystać w FORTH możliwości graficzne i dźwiękowe komputera, należy skorzystać z owych dodatkowych pakietów albo przy pewnej znajomości mapy pamięci komputera posłużyć się jego systemem operacyjnym i ewentualnie zbudować nowe słowa. Sposób korzystania w FORTH z grafiki i dźwięku na Atari 4OO/8OOXL i 130XE przedstawiamy w załączniku.

Uderzającą właściwością języka FORTH jest to, że wszystkie narzędzia programistyczne, o których mowa, są objętościowo bardzo małe, mikroskopijne wprost w porównaniu z możliwościami, jakie zapewniają.

Interpretator wraz z podstawowym słownikiem FORTH-a zajmuje, zależnie od implementacji, 7-8 kilobajtów, edytor niespełna kilobajt, a asembler ok. 1,5 KB. Parę kilobajtów zabierają inne składniki systemu, zwłaszcza bufory. W 64-kilobajtowym komputerze, takim jak Atari czy Commodore, po wprowadzeniu edytora, asemblera i innych uzupełnień (niekoniecznie wszystkich naraz) zostaje jeszcze co najmniej 35 kilobajtów wolnej pamięci. Jest to dla pamięciooszczędnego FORTH-a ilość ogromną.

C Z Ę Ś Ć I

PODSTAWY

ROZDZIAŁ 2
PIERWSZE KROKI

2.1 Laik przy komputerze

Pierwsze kroki w poznawaniu nowego języka programowania są zawsze trudne. Przypomnijmy sobie, jak to było z naszym startem w BASIC-u czy LOGO.

Przypuśćmy, że zupełnie nie znamy FORTH-a. Od czego zacząć? Oczywiście, najpierw trzeba wgrać do pamięci komputera interpretator. Nie smućmy się, jeżeli nie zawiera wszystkich programów systemowych, o których była mowa. Na początek, do zawarcia pierwszej znajomości, wystarczy podstawowy słownik FORTH. Korzystanie z edytora wymaga pewnej wprawy. Jeżeli nie mamy stacji dyskietek, jest on zresztą mało przydatny.

Interpretator został wgrany. Na ekranie pojawia się zwykle napis informujący o nazwie, cechach i dacie powstania interpretatora. Zawsze zaś zobaczymy napis: "ok" - skrót angielskiego słowa "okey" - "w porządku". Napis ten oznacza, że FORTH gotów jest przyjąć i wykonać nasze polecenia.

Napiszmy jakąś liczbę, powiedzmy 7, i naciśnijmy klawisz RETURN. FORTH odpowiada "ok", ale właściwie nie wiemy, co się stało. Napiszmy kropkę (Return) . FORTH pisze: "7 ok". Pora to wyjaśnić. Gdy napisaliśmy 7, liczba ta została wprowadzona na stos. natomiast kropka, która, choć brzmi to zabawnie, jest słowem FORTH-a, służy do tego, by usuwać liczby ze szczytu stosu i drukować je. Poznaliśmy zatem bardzo często używane słowo, ważne, choć wygląda tak niepozornie. Wiele zresztą, zwłaszcza często używanych, słów , ma nazwy proste i krótkie, nieraz jednoznakowe. Słowami są dwukropek, średnik, wykrzyknik, apostrof.

Słowem jest także przecinek, który w FORTH nie pełni roli znaku rozdzielającego liczby czy słowa. W FORTH pod-

stawowym separatorem jest s p a c j a czyli puste miejsce powstające po naciśnięciu dolnego wydłużonego klawisza. Użytkownicy BASIC-u muszą się do tego przyzwyczaić, znający LOGO nie będą mieli kłopotu, bo w nim jest tak samo. W FORTH każde słowo, każdy znak, w tym znaki działań arytmetycznych, musi być oddzielony od sąsiedniego co najmniej jedną spacją. Może ich być więcej.

Napiszmy teraz jakiś wyraz, powiedzmy KOT (Return) . FORTH powtarza ten wyraz ze znakiem zapytania. Oznacza to, że nas nie zrozumiał, a stało się tak dlatego, że nasze wejście było niepoprawne.

Zróbmy kolejny eksperyment. Napiszmy dwukropek, spacja, KOT (Return) . Zwróćmy uwagę, że tym razem nie pojawił się napis "ok". Dlaczego? Ponieważ dwukropek z obowiązkową po nim spacją jest jednym z najważniejszych słów kluczowych. Służy mianowicie do definiowania nowych słów. Przykład jego. zastosowania znaleźliśmy już na poprzednich stronach. Pisząc

: KOT

zawiadomiliśmy FORTH, że zamierzamy zdefiniować słowo o n a z w i e KOT. Zapamiętajmy, że słowa FORTH-a mają nazwy. Czym innym jest słowo, a czym innym jego nazwa. Ta ostatnia powoduje, że uruchomione zostają czynności przewidziane w słowie, np. nazwa . (kropka) uruchamia czynność wydrukowania liczby ze szczytu stosu.

No dobrze, ale co teraz robić? FORTH przecież na coś czeka. Postawmy w nowym wierszu średnik i naciśnijmy Return. Znowu pojawia się znany napis "ok". Co się zmieniło? Pisząc słowo ; czyli średnik, oznajmiliśmy, że definiowanie słowa zostało zakończone. Możemy je teraz napisać i FORTH odpowie "ok". Jednakże słowo KOT w rzeczywistości nie będzie robiło nic, jest bowiem słowem pustym. Zawiera nazwę, ale nie określa czynności, które ma wywołać. Napiszmy teraz:

FORGET KOT { Return)

FORGET powoduje, że definicja słowa zostaje zapomniana, usunięta ze słowniku, zresztą - zapamiętajmy to - wraz z

wszystkimi definicjami słów, które stworzylibyśmy po niej. Napisanie KOT znów wywoła sygnał o błędzie. Napiszmy teraz:

```
:KOT CR  ." Kot pije mleko" ; (Return)
```

FORTH odpowiedział "ok", a do słownika weszło znów słowo KOT. Możemy się o tym przekonać pisząc użyteczne słowo VLIST. Powoduje ono wyświetlenie nazw wszystkich słów znajdujących się w słowniku w odwrotnej kolejności; od najpóźniej do wcześniej zdefiniowanych. Słowo KOT wyświetlone zostanie na samym początku.
Napiszmy KOT (Return) . FORTH drukuje: Kot pije mleko ok. W tym i dalszych przykładach książki podkreślenie oznaczać będzie, że tekst napisał komputer. Pomijać będziemy na ogół zaznaczenie użycia klawisza Return. FORTH z reguły po naciśnięciu tego klawisza pisze w tym samym wierszu, jeżeli nie spowodujemy innej formy wydruku. Miejsce, gdzie zaczyna się tekst podkreślony, będzie zatem miejsce użycia klawisza Return.
Z pomocą słowa CR można spowodować, rozpoczęcie druku od nowego wiersza.
Dlaczego wydrukowane zostało zdanie: Kot pije mleko? Stało się tak dlatego, że powtórnie zdefiniowane słowo KOT nie jest już puste, związana jest z nim pewna czynność. Para znaków ." jest też słowem. Powoduje ono, z obowiązkową po nim spacją, wydrukowanie dalszego tekstu aż do końcowego, cudzysłowu. Nasze słowo KOT wykonało tę czynność. Końcowy cudzysłów należy, podobnie jak w BASIC-u, stawiać bezpośrednio za ostatnim znakiem przeznaczonym do wydrukowania. Znakiem tym może być również spacja w celu stworzenia odstępu między słowami.
FORTH rozporządza dwu słowami służącymi do tworzenia odstępów. SPACE powoduje, że wprowadzona zostaje jedna spacja. Natomiast słowo SPACES wymaga określenia liczby spacji. Liczbę tę podajemy przed nazwą słowa. Na przykład

```
7 SPACES
```

spowoduje wydrukowanie siedmiu pustych odstępów.

2.2 Poznajemy pętlę

W przykładzie służącym porównaniu szybkości działania BASIC-u i FORTH-a użyliśmy słów DO ... LOOP. Jest to jedna z ważnych konstrukcji programowania strukturalnego w FORTH, które będą szczegółowo omówione w rozdziale 5. Wyjaśnijmy jej budowę i działanie. Jest to tak zwana pętla liczona. Z podanego przykładu możemy się domyśleć, że działa tak samo jak FOR ... NEXT w BASIC-u. Natomiast forma zapisu jest inna i można ją przedstawić następująco:

<granica-górna+1> <granica-dolna> DO <czynność> LOOP

Tak więc wartości graniczne wskaźnika pętli piszemy przed jej początkiem, który stanowi słowo DO. Istnieje także słowo I - wskaźnik pętli, który możemy wykorzystać.

```
: TEST2 11 0 DO I . LOOP ;
```

Słowo TEST2 wydrukuje po jego wywołaniu, czyli napisaniu go, liczby od 0 do 10, a nie do 11, pamiętajmy o tym. Konstrukcja DO ... LOOP należy do stosunkowo nielicznych, które można stosować tylko wewnątrz definicji. Próbą użycia jej wprost z klawiatury spowoduje sygnał o błędzie, Wskaźnikiem pętli może być tylko I, a nie dowolna zmienna, jak to się dzieje w BASIC-u. Jeszcze przykład

```
: TEST3  7  0  DO  ." A "  LOOP ;
```

TEST3 wydrukuje rząd siedmiu A.

2.3 Podstawowe reguły dialogu

Czas uogólnić wyniki dotychczasowych prób. Jak reaguje interpretator FORTH-a na wszystko to, co wprowadzamy z klawiatury? Dokonuje on natychmiast po naciśnięciu klawisza Return kontroli wprowadzonej linii i postępuje następująco:

	RODZAJ WPROWADZENIA	DZIAŁANIE
1.	Słowo zdefiniowane	Wykonuje natychmiast
2.	Liczba	Umieszcza na stosie
3.	Nowe definiowane słowo	Kompiluje i wprowadza do słownika
4.	Niepoprawne wejście	Sygnalizuje błąd i opróżnia stos

W poprzednich przykładach poznaliśmy wszystkie te sytuacje, a tym samym - najważniejsze zasady działania FORTH-a. Jeszcze jedno wymaga wyjaśnienia. Zauważyliśmy wyraźną różnicę między zachowaniem FORTH-a w sytuacjach 1 i 2, a jego. zachowaniem w sytuacji 3. Różnica polega na tym, że podczas definiowania nowego słowa FORTH zawiesza wykonywanie bieżących poleceń, natomiast opracowuje wraz z nami definicję nowego słowa, bacznie przy tym czuwając nad poprawnością udzielanych przez nas poleceń. Sprawdza między innymi, czy nie użyliśmy w definicji słowa dotychczas nie zdefiniowanego oraz czy poprawnie stosujemy słowa kluczowe.

Przypomnieć tu należy wcześniej omówioną sprawę dwóch faz działania interpretatora: fazy kompilacji i fazy wykonywania.

Zetknęliśmy się dotychczas z najważniejszą, choć nie jedyną, metodą definiowania nowych słów z pomocą tzw. definicji dwukropkowej (colon definition). To właśnie dwukropek wraz z następującą po nim nazwą słowa powoduje przejście w fazę kompilacji. Podczas tej fazy ustaje bieżące wykonywanie poleceń.

Przestaje działać zasada, że po naciśnięciu klawisza Return następuje natychmiastowe wykonanie wszystkich słów i poleceń wypisanych w linii. Możemy teraz dowolnie posługiwać się Returnem, rozmieszczać składniki słowa w sposób przejrzysty, z dużymi odstępami. To, co napiszemy na ekranie, zaraz, potem zniknie, bowiem w FORTH nie ma listingu w klasycznym znaczeniu tego słowa. Natomiast dbałość o graficzną estetykę definiowanych słów pożądana jest przy pracy z edytorem, bowiem po wpisaniu na dyskietkę kolejnego ekranu słowa zachowują układ, jaki im nadaliśmy.

Uprzedzając informacje, które potem nastąpią, stwierdźmy, że nazwę ekranu nosi w FORTH podstawowa jednostka gromadzenia danych oraz ich wymiany między edytorem, a stacją dyskietek. Na ogół współcześnie wynosi ona 1024 bajty, choć zdarzają się ekrany mniejsze. Jest także pojęcie bloku, który w fig-FORTH liczy 128 bajtów, czyli tyle samo, co sektor dyskietki. Wielkości właściwe dla danego systemu można odczytać pod adresami stałych B/SCR i B/BUF. Pisząc

B/SCR . Return

uzyskujemy informację, ile bloków mieści się w ekranie a B/BUF podaje liczbę bajtów w bloku. Długość linii podaje stała C/L (napiszmy C/L.), wynosi ona zwykle 64. lecz czasem 32 bajty. Różnice tych wielkości w rozmaitych interpretatorach mogą utrudnić przenoszenie programów.
Zamykając rozdział przypomnijmy, ze poznaliśmy w nim następujące słowa FORTH-a:

. " ," " : ; SPACES VLIST B/BUF B/SCR C/L CR FORGET SPACE

Ćwiczenie 1

Używając zdefiniowanego już słowa KOT stwórzmy z jego pomocą i ewentualnie z pomocą SPACE nowe słowo, które drukować będzie napis: Kot pije mleko i poluje na myszy.

ROZDZIAŁ 3
W ŚWIECIE LICZB

3.1 Rodzaje liczb

FORTH rozporządza szerokimi możliwościami stosowania w obliczeniach rozmaitych typów liczb. Dlaczego jest to ważne? Przede wszystkim dlatego, że im mniej bajtów zajmuje liczba, tym większa staje się szybkość obliczeń.
W podstawowych wersjach wielu języków, takich jak BASIC czy LOGO, użytkownik ma nikłe możliwości wyboru dogodnego dlań typu liczb. Np. liczba całkowita 10 mieszcząca się swobodnie w jednym bajcie traktowana jest w tych językach tak samo jak 1000000 i zajmuje na ogół sześć lub siedem bajtów, co można uznać za niemałą rozrzutność.
W FORTH istnieje rozbudowany mechanizm pozwalający z pomocą odpowiednich zestawów operatorów na posługiwanie się liczbami o najdogodniejszej w danej chwili długości, przekształcanie liczb z jednego typu na inny itd.
Oto podstawowe rodzaje stosowanych liczb:

A. pojedynczej długości ze znakiem. Liczby te zajmują w pamięci komputera dwa bajty, przy czym najwyższy bit górnego bajtu przeznaczony jest na znak. Tym samym liczby te mogą przybierać wartości od -32768 do +32767;

B. pojedynczej długości bez znaku. Zajmują dwa bajty i mogą przybierać wartości od zera do 65535;

C. podwójnej długości ze znakiem. Zajmują cztery bajty, przy czym najstarszy bit przeznaczony jest na znak. Pozwala to na stosowanie liczb w granicach od -2147483648 do 2147483647 czyli przeszło dwóch miliardów poniżej i powyżej zera.

D. podwójnej długości bez znaku. Zajmują cztery bajty i mogą przybierać wartości od zera do 4294967295 czyli do przeszło czterech miliardów;

E. zmiennopozycyjne. Forma przedstawiania tych liczb w pamięci jest zupełnie inna; w postaci znaku oraz wykładnika potęgi i mantysy;

F. wartość kodu ASCII. Zajmują jeden bajt.

Jakkolwiek w FORTH możliwe jest stonowanie arytmetyki zmiennopozycyjnej, jest on zorientowany przede wszystkim na posługiwanie się liczbami całkowitymi pojedynczej i podwójnej długości. Określa się je również jako liczby pojedynczej i podwójnej dokładności lub po prostu jako pojedyncze i podwójne. Stosowanie tych typów liczb pozwala osiągnąć wysoką szybkość obliczeń, a więc również skraca czas wykonywania wszelkich programów. Oczywiście, narzucone są przy tym granice zakresów liczb, a ich przekroczenie powoduje powstawanie błędnych wyników.

Nie zawsze są to ograniczenia uciążliwe. Np. z pomocą liczb pojedynczej długości bez znaku można przedstawić wszystkie adresy w komputerze, wielkości niezbędne przy generowaniu obrazu i dźwięku oraz, oczywiście, znaki w kodzie ASCII i wartości logiczne. Tym samym już przy ograniczeniu się do tego typu liczb można wykonać bardzo wiele zadań programistycznych, w tym prawie wszystkie związane z projektowaniem gier.

Liczby podwójnej długości rozszerzają zakres liczb jeszcze bardziej. Dzięki tym możliwościom arytmetyki stałopozycyjnej FORTH może być z powodzeniem stosowany do przetwarzania tekstów, gromadzenia i przetwarzania danych z urządzeń pomiarowych w przemyśle oraz wielu innych zadań technicznych, a także ekonomicznych, w tym ze strefy zarządzania.

Każde zwiększenie długości liczb o bajt powoduje znaczne, w przybliżeniu dwukrotne, wydłużenie się czasu wykonania. W niejednym zatem przypadku opłaca się dołożyć wysiłku, by przystosować dane do możliwości obliczeń stałopozycyjnych, całkowitoliczbowych.

W ciągu lat rozwoju techniki komputerowej całe działy matematyki, jak kombinatoryka, teoria grafów, algebra komputerowa, rachunek zbiorów, teoria automatów przeżywały I

przeżywają gwałtowny i szybki rozwój. Jednym z najważniejszych kierunków ich dociekań jest poszukiwanie dróg zwiększenia efektywności i skrócenia czasu wykonania algorytmów. Nie jest przy tym przypadkiem, że często pada pytanie: jak rozszerzyć zakres stosowania liczb całkowitych? Warto o tym pamiętać i doskonalić umiejętności wykorzystania instrumentów szybkich obliczeń, jakimi rozporządza FORTH. W języku tym istnieją zarazem środki pozwalające przedstawiać liczby w postaci ułamków dziesiętnych

Istnieją zestawy słów do wykonywania obliczeń na różnych typach liczb. Te, które służą do operacji na liczbach podwójnych, mają zazwyczaj nazwy z przedrostkiem "2" lub "D", Przedrostek "U" wyróżnia operatory dla działań na liczbach bez znaku, a "M" - dla działań z liczbami "mieszanej", pojedynczej i podwójnej długości.

W celu wyróżnienia w komentarzach do programów poszczególnych typów liczb stosować będziemy jednolicie następujące, wywodzące się z angielskich nazw oznaczenia.

n - liczba pojedynczej długości ze znakiem

u - " " " bez znaku

d - " podwójnej " ze znakiem

ud - " " " bez znaku

fl - liczba zmiennopozycyjna - floating

c - znak w kodzie ASCII.

3.2 Potęga stosów

Jednym z najciekawszych pomysłów wynalazcy FORTH-a było szerokie zastosowanie w nim organizacji pamięci zwanej stosem.

Czym jest stos? Jest to fragment pamięci operacyjnej (tzn. bieżąco wykorzystywanej) komputera przeznaczony do przechowywania danych, a odznaczający się szczególną organizacją: dostępny jest tylko obiekt, który był wprowadzony na stos jako ostatni. Mówimy o nim, że znajduje się na szczycie stosu. Aby dotrzeć do wcześniej wprowadzonego obiektu, trzeba przedtem usunąć te, które wprowadziliśmy po

nim. Ważnym składnikiem w tej strukturze jest wskaźnik stosu. Podaje on adres, pod którym znajduje się pierwsza wolna komórka stosu. W FORTH bieżącą wartość wskaźnika stosu podaje zmienna SP@ .

Stos można porównać do talii kart położonej na stole koszulkami do dołu. Patrząc na nią widzimy tylko jedną kartę, leżącą na szczycie talii. Dopiero zdejmując kolejne karty zobaczymy następne. Kładąc z powrotem karty pojedynczo w dowolnej kolejności, kładziemy je zawsze na wierzch talii, a potem zdejmujemy w kolejności odwrotnej w stosunku do tej, w jakiej je po raz ostatni położyliśmy. Inne porównanie: stos działa podobnie jak magazynek pistoletu: naboje wkładać i wyjmować można tylko z jednego, otwartego końca.

Taka zasada działania stosu określana jest po angielsku mianem LIPO - Last In First Out (ostatni wszedł, pierwszy wyszedł) .

Stosowa organizacja pamięci znana jest od dawna i szeroko wykorzystywana do różnych celów. Np. wbudowany stos komputera w przypadku skoku GOSUB w BASIC-u przechowuje adres spod którego nastąpił skok, aby po komendzie RETURN powrócić we właściwe miejsce.

Jednakże twórca FORTH-a posłużył się stosem w niezmierni oryginalny sposób: wykorzystał go jako ogniwo pośrednie wszystkich operacji arytmetycznych i logicznych. Zawsze wtedy, gdy dokonujemy w FORTH jakiejkolwiek czynności, w której trzeba się posłużyć liczbą, musi być ona brana ze stosu, czyli wcześniej tam wprowadzona.

FORTH ma dwa stosy. Stos główny - to struktura, niezmiernie szeroko wykorzystywana, do której użytkownik ma łatwy bezpośredni dostęp. Drugi stos nosi nazwę stosu powrotów (return stack) , a to dlatego, że jednym z jego zastosowań jest przechowywanie danych służących do powrotów z podprocedur, do kontroli wykonania pętli i do innych celów związanych z funkcjonowaniem systemu. Stos ten jest również dostępny dla użytkownika, trzeba to jednak czynić ostrożnie i umiejętnie. Do szczegółowego omówienia działania stosów i służących do tego słów wrócimy niebawem w tym rozdziale.

3.3 Odwrotna notacja polska (ONP)

Sprawi nam bez wątpienia satysfakcję, że sposób zapisu wyrażeń arytmetycznych stosowany w FORTH ma w swej nazwie przymiotnik "polski". Ów odmienny od potocznego sposób umieszczania operatorów, np. znaków działań, po operatach, czyli wartościach, a nie między nimi jest zmodyfikowaną wersją notacji wynalezionej przez wybitnego logika polskiego Jana Łukasiewicza (1878-1956) . Łukasiewicz umieszczał symbole operatorów przed symbolami operandów, np. zamiast a+b pisał + a b.

W odwrotnej notacji napiszemy a b +.

Odwrotna notacja polska (ONP) , znana pod tą nazwą w całym świecie, stała się bardzo popularna w szeregu zastosowań, np. w wielu kalkulatorach, a także przy translacji wyrażeń arytmetycznych języków programowania automatycznego.

Jak przedstawia się jej stosowanie w FORTH? Wyjaśnijmy to z pomocą przykładów. Po lewej wyrażenia w postaci "zwykłej", po prawej ich odpowiedniki w FORTH.

2 + 3	2 3 +
(2 + 3)* 7	2 3 + 7 *
(5 - 7)*(10 + 8)	5 7 - 10 8 + *

wróćmy uwagę na ostatni przykład. Układ operatorów jest tu inny niż w zapisie z nawiasami: działania wykonujemy w kolejności ich pierwszeństwa.

Skoro użyliśmy operatorów arytmetycznych należy powiedzieć, jak działają. Gdy napiszemy 2 3 + , 2 i 3 znikną ze stosu i zostaną z a s t ą p i o n e przez wynik: 5. Jest to reguła przy wszystkich operacjach na stosie.

Trochę niecodzienna forma zapisu sprawiać może na początku pewne trudności, natomiast zaletą odwrotnej notacji polskiej jest m. in. to, że eliminuje potrzebę używania nawiasów. W FORTH jej stosowanie jest naturalną konsekwencją faktu, że w działaniach możemy się posłużyć tylko liczbami obecnymi na stosie. Najpierw więc trzeba wprowadzić na stos 2 i 3, a potem znakiem + poinformować FORTH, że chcemy je dodać.

Odwrotna notacja polska stosowana jest nie tylko w działaniach arytmetycznych. Każda liczba musi się znaleźć na stosie przed jej wykorzystaniem, tzn. musi być napisana przed słowem (operatorem) określającym czynność. Widzieliśmy to już na przykładzie zastosowania słowa SPACES. Jeżeli napiszemy odwrotnie, a zgodnie z przyzwyczajeniami wdrożonym: przez BASIC SPACES 7, to nastąpi sygnał o błędzie. Słowo SPACES oczekuje liczby na stosie, ale nie znajduje jej, bowiem stos jest pusty.

FORTH ma szczególny sposób sygnalizowania błędów, pobiorą mianowicie ich definicje z odpowiednich ekranów, zwykle 4 i 5, dyskietki lub z buforu wprowadzania z kasety. Gdy ich tam nie zastaje, "robi szum"" sygnalizuje, że jest źle, ale nie może nam wyjaśnić, na czym polega błąd. Musimy się wtedy sami domyśleć przyczyny.

W omawianym przykładzie chodzi o szczególny przypadek szerszej kategorii najczęstszych błędów polegających na tym, że na stosie jest za mało lub za dużo liczb w stosunku do potrzeb wykonywanego programu. Nadmiar jest także szkodliwy, bowiem stosy są stosunkowo nieduże, mogą zazwyczaj pomieścić 20-40 liczb pojedynczej długości, a przepełnienie stosu - to najczęściej załamanie się programu. Dlatego stawiając pierwsze kroki w FORTH najpierw poćwiczmy działania arytmetyczne. Pamiętajmy, że liczby i znaki zawsze muszą być rozdzielone spacjami.

Ćwiczenie 2

Zapiszmy w FORTH następujące wyrażenia:

```
(A+B)*(C-D)

 A* (B-C )+D

(A+B)/C+(D+E)*F

A/(B+C )*(D+A)*(C+D)
```

Ćwiczenie 3

Jaki będzie wynik następujących działań?

a)7 5 * 3 + 2 / 4 - 7 /
b)b) 22 3 5 + / 7 10 +*
c)C) 17 34 + 3 / +

3.4 Manipulowanie liczbami na stosie

3.4.1 Budowa stosu

Stos główny - to podstawowe ogniwo pośrednie w operowaniu wszelkimi wartościami. Pod pojęciem wartości rozumiemy nie tylko liczby używane w operacjach arytmetycznych przy wykonywaniu słów, lecz także wartości logiczne, które zresztą mają w FORTH postać liczbową oraz wartości kodu ASCII odpowiadające poszczególnym znakom.

Stos główny, podobnie jak stos powrotów, zorganizowany jest jako ciąg komórek dwubajtowych. Liczba pojedynczej dokładności zajmuje jedną taką komórkę, podwójna dwie.

3.4.2 Wprowadzanie i wyprowadzanie liczb

Wprowadzanie na stos liczby pojedynczej długości polega po prostu na napisaniu liczby i naciśnięciu Returnu.
Liczbę podwójnej długości interpretator poznaje po tym, że w dowolnym miejscu, na początku, w środku lub na końcu zawiera ona kropkę. Obecność tej kropki powoduje, że FORTH dla liczby na stosie rezerwuje dwie komórki dwubajtowe zamiast jednej.

Istnieją trzy odrębne słowa do wyprowadzania liczb ze stosu i ich drukowania.

. - pobiera i drukuje liczbę pojedynczej długości ze znakiem

U. - pobiera i drukuje liczbę pojedynczej długości bez znaku

D. - pobiera i drukuje liczbę podwójnej długości.

Przykłady:

```
40000 ok
. -25536 ok
40000 U. 40000 ok
1000000 .  16960 ok
1000000   . . 15 16970 ok
1000000. ok
D. 1000000 ok
```

Przykłady te wskazują, że w celu poprawnego wydrukowania liczby pojedynczej długości bez znaku większej niż 32767 leży stonować U., a dla liczb podwójnych ze znakiem D. - słowo drukujące takie liczby.

FORTH zawiera dwa wygodne słowa pozwalające wyprowadzić liczby na ekran lub drukarkę w sposób przejrzysty i estetyczny.

```
.R  długość-pola n ---
D.R długość-pola d ---
```

Słowa te, jak już o tym wspomniano, powodują, że liczby nie są drukowane jak zwykle, z odstępem jedynie spacji, lecz umieszczane w polach o ustalonej przez nas długości i dosunięte do prawych krawędzi tych pól. Dzięki temu cyfry jednostek mogą się znaleźć pod jednostkami, dziesiątków pod dziesiątkami itd.

Przykład:

```
: TEST 7 333 55 9999 20000 CR
5 0 DO 6 .R CR LOOP ;
```

da wydruk:

```
20000
 9999
   55
  333
    7
```

W słowie TEST najpierw wprowadziliśmy pięć liczb. Potem podane zostały granice pętli (5 0) i wreszcie DO... LOOP wydrukowało liczby, każdą od nowego wiersza, przy czym wskaźnik 6 przy .R spowodował, że cyfry jednostek znalazły się w szóstej kolumnie od lewego brzegu ekranu.

Z pomocą obu słów można uniknąć tego, że część znaków przy końcu linii ekranowej zostaje przeniesiona do następnej linii. Wystarczy rozmierzyć ekran na równe odcinki, np. po 8 znaków. Część komputerów ma ponadto funkcje zmiany lewego i prawego marginesu.

3.4.3 Przestawienia

Znaczenie stosu głównego jako pośrednika w operowaniu liczbami skłoniło Moore'a do pewnego odstępstwa od ogólnej zasady, wedle której do stosu można uzyskać dostęp tylko z jego szczytu. Z pomocą szeregu słów możemy dokonywać na stosie rozmaitych manipulacji. Przypomina to, jak zobaczymy, rozwiązywanie łamigłówek i na pierwszy rzut oka mogłoby się wydawać mało użyteczne, w rzeczywistości słowa manipulujące stosem należą w FORTH do najczęściej używanych i bardzo usprawniają działanie programów.

Oto słowa wykonujące manipulacje z liczbami pojedynczej długości ze znakiem i bez znaku :

DUP tworzy kopię liczby znajdującej się na szczycie stosu i umieszcza ją nad nią;

DROP usuwa liczbę ze szczytu stosu, ale nie drukuje jej, jak słowo kropka;

SWAP zamienia miejscami dwie ostatnio wprowadzone liczby?

OVER tworzy kopię drugiej liczby na stosie i umieszcza tę kopię na szczycie stosu nie zmieniając pozostałych;

ROT przestawia trzy pierwsze liczby w ten sposób, że trzecią umieszcza na szczycie stosu, a dwie pozostałe obniża o miejsce.

Uzupełnieniem tego zestawu jest słowo -DUP, w niektórych systemach ?DUP. Kopiuje ono liczbę tylko wtedy, gdy nie jest ona równa zeru.

Waga stosu i obfitość manewrów, które można na nim wykonać, czynią nieraz potrzebnym poznanie aktualnego stanu stosu bez naruszania jego zawartości.

Poniższe bardzo użyteczne słowo wyświetla stos od lewej do prawej, tak iż wartość na szczycie jest ostatnią po prawej. Zdefiniujemy je bez wyjaśniania jego działania.

```
: .S 2 SPACES SP@ S0 @ = 0=
     IF
       SP@ 2 - SO @ - DO I @ . -2 +LOOP
     THEN ;
```

Słowo 0= jest w tym wypadku równoważne operatorowi logicznemu NOT i może być przez niego zastąpiono. Nie we wszystkich implementacjach istnieje SO, które jest zmienną zawierającą początkową wartość wskaźnika stosu SP@. Można wówczas postąpić następująco: przy pustym stosie ustalamy wartość wskaźnika stosu pisząc SP@ i kropkę. Następnie liczbą która zostanie przy tym wydrukowana, zastępujemy w dwóch miejscach definicji pary słów: SO @.

Wypróbujmy nasze nowe słowo:

```
1 2 3

.S 1 2 3 ok
```

Tekst drukowany przez komputer podkreślamy. A oto dalsze próby potwierdzające działanie słów zmieniających układ liczb na stosie:

```
DUP  .S 1 2 3 3 ok
DROP .S 1 2 3 ok
SWAP .S 1 3 2 ok
ROT  .S 3 2 1 ok
OVER .S 3 2 1 2 ok
```

A teraz dwa proste przykłady zastosowań nowo poznanych słów w definicjach.

```
: SZESCIAN  DUP  OVER  *  *  .  ;

: KWADRAT  DUP  *  .  ;
```

Pierwsze słowo wydrukuje trzecią potęgę, a drugie kwadrat liczby. A oto program, który wydrukuje na ekranie serię kwadratów liczb "w słupku":

```
: KWADRATY  CR  DO  I  DUP  U*  15  D.R  CR  LOOP  ;
```

Pisząc teraz

```
KWADRATY 45001  41980
```

otrzymamy wydruk dziesięciu kwadratów liczb;

```
2023200400
2023290361
. . . . . .
2024910001
2025000000
```

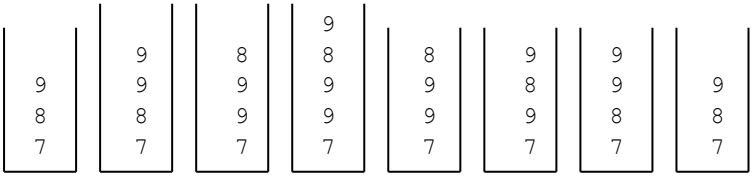
W słowie tym, obok znanej już pętli liczonej zastosowane zostało U* - nie poznane jeszcze, a dające wynik mnożenia w postaci liczby podwójnej precyzji.

3.4.4 Jak kontrolować stos?

Użyteczne jest poznawanie stanu stosu w trakcie wykonywania programu, ale jaszcze ważniejsze - w fazie projektowania. Istnieją dwie proste metody kontrolowania stosu w tej fazie, Które warto stosować. Jedną jest korzystanie z diagramów, w które wpisujemy kolejne stany stosu po każdym kroku. Druga - to podawanie w nawiasie obok słowa stanu szczytu stosu przed i po wykonaniu słowa, przy czym miejsce słowa w tym ciągu zaznaczamy z pomocą --- . Tę drugą metodę będziemy w książce często stosować.

Oto przykład sekwencji czynności ilustrowanej diagramami:

7 8 9 DUP ROT OVER DROP ROT ROT DROP



Stosując drugą z metod możemy następująco przedstawić działanie ostatnio poznanych słów.

DUP (n --- n n)

DROP (n ---)

SWAP (n1 n2 --- n2 n1)

OVER (n1 n2 --- n1 n2 n1)

ROT (n1 n2 n3 --- n2 n3 n1)

Kolejny zestaw słów umożliwia manipulowanie na stosie liczbami podwójnej długości.

2DUP (d --- d d)

2DROP (d ---)

2SWAP (d1 d2 --- d2 d1)

2OVER (d1 d2 --- d1 d2 d1)

2ROT (d1 d2 d3 --- d2 d3 d1)

Nie wszystkie interpretatory FORTH-a zawierają te słowa. Możemy jednak łatwo sami je zdefiniować. Wystarczy pamiętać że każda liczba podwójna przedstawiona jest na stosie z pomocą pary liczb pojedynczej długości i manipulować tymi ostatnimi. Omówimy to w następnym punkcie.

Na zakończenie wstępnych informacji o stosie głównym wskażmy na dwa ważne słowa związane z jego działaniem. Poznane SP@ pozwala ustalić adres pierwszej wolnej komórki stosu. SP! inicjalizuje, czyli opróżnia stos bez ubocznych skutków. Jak już mówiliśmy, każde błędne wejście także opróżnia stos, jednak wywołuje sygnał o błędzie.

Omówione tu zostały podstawowe i najpowszechniej stosowane słowa do manipulowania stosem. FORTH-83 zalicza do obowiązkowego standardu dwa jeszcze słowa o bardziej skomplikowanym działaniu: PICK i ROLL, Ich znaczenie podajemy w leksykonie zamieszczonym w aneksie. W praktyce słowa te nie są zbyt użyteczne, zwłaszcza ROLL jest bardzo powolne.

Ćwiczenie 4

Przedstawmy przy użyciu nawiasów kolejne fazy wykonywania poniższych czynności. Pamiętajmy, że przy działaniach arytmetycznych operandy zostają zastąpione przez wynik,

```
9  11  13  15  *  ROT  -  /
```

3.5 Stos powrotów

W przeciwieństwie do stosu głównego stos powrotów jest w poważnej mierze chroniony przed ingerencjami użytkownika. Uczyniono tak dlatego, że spełnia on rolę ważnego składnika funkcjonowania FORTH-a, jego wewnętrznej struktury. Do stosu powrotów nie ma innego dostępu, aniżeli przez szczyt. Nie można na stosie powrotów stosować żadnych przestawień, takich jak na stosie głównym.

Mimo tych ograniczeń stos powrotów jest wysoce użyteczny dla użytkownika dzięki możliwości korzystania z niego z pomocą niedużego zestawu słów zapewniających komunikację między obu stosami FORTH-a.

R>	(--- n)	zabiera liczbę ze szczytu stosu głównego i umieszcza ją na szczycie stosu powrotów
>R	(n ---)	dokonuje przemieszczenia w odwrotnym kierunku, ze stosu powrotów na główny
R	(--- n)	kopiuje na stos główny szczytową wartość stosu powrotów, pozostawia ją jednak na tym ostatnim.

Uwaga!

Ważną zasadą przy korzystaniu ze stosu powrotów jest

to, że trzeba go pozostawić w takim samym stanie, w jakim był, zanim z niego skorzystaliśmy w bezpośrednim wykonaniu lub w jakimkolwiek skompilowanym słowie. Błąd z reguły prowadzi do unieruchomienia interpretatora. Zastosowane ograniczenie służy bezpieczeństwu programów, bowiem na stosie powrotów nogą znajdować się również wartości decydujące o działaniu pętli, przejść, rozwidleń itd.

Stos powrotów możemy wykorzystać jako miejsce krótkotrwałego (bardzo krótkotrwałego!) przechowywania wartości ze stosu głównego. Czasem chcemy, na przykład, dotrzeć do wartości położonej głębiej na stosie głównym, gdy tymczasem standardowe słowo ROT sięga najdalej do trzeciej komórki. Przenosząc na krótko dwie górne liczby na stos powrotów uzyskujemy Już dostęp do piątej komórki.

W ten właśnie sposób możemy z pomocą stosu powrotów zbudować definicje dla manipulowania liczbami podwójnymi.

```
: 2DUP  OVER  OVER  ;

: 2DROP  DROP  DROP  ;

: 2SWAP  ROT  >R  ROT  R>  ;

: 2OVER  >R  >R  2DUP  R>  R>  2SWAP  ;

: 2 ROT  >R  >R  2SWAP  R>  R>  2SWAP  ;
```

Uważniej prześledźmy ten mechanizm przy najbardziej złożonym słowie 2ROT. Wymaga ono manewrowania trzema liczbami podwójnymi czyli aż sześcioma komórkami dla liczb pojedynczej długości. Przedstawmy kolejne fazy definicji przy użyciu wspomnianych już oznaczeń w nawiasach. Stan początkowy i końcowy wyrazimy z pomocą liczb pojedynczej długości.

```
: 2ROT  ( n1 n2 n3 n4 n5 n6 --- n3 n4 n5 n6 n1 n2 )
>R (n1 n2 n3 n4 n5 )( przenosi n6 na stos powrotów )
>R ( n1 n2 n3 n4 ) ( przenosi n5 na stos powrotów )
2SWAP ( n3 n4 n1 n2 )  ( zamienia pary liczb )
R>  ( n3 n4 n1 n2 n5 ) ( przenosi z powrotem n5 )
R>  ( n3 n4 n1 n2 n5 n6 )( przenosi z powrotem n6 )
2SWAP (n3 n4 n5 n6 n1 n2 )  ( zamiana wykonana! )
```

Ćwiczenie 5

Zdefiniujmy słowo 3DUP, które skopiuje trzy kolejne liczby z góry stosu i w takiej samej kolejności umieśćmy je na szczycie. Jest to możliwe do wykonania w 7 krokach.

Ćwiczenie 6

Ustawmy 6 górnych liczb na stosie w odwrotnej kolejności. Jest to możliwe w 17 krokach.

3.6 Arytmetyka udoskonalona

W stosunku do tradycyjnego zestawu operacji arytmetycznych FORTH oferuje znacznie bogatszą gamę możliwości, które kolejno omówimy.

3.6.1 Liczby pojedynczej długości

Trzy z czterech podstawowych operatorów działań funkcjonują wedle powszechnie znanych zasad, natomiast w przypadku dzielenia należy pamiętać, że jest ono dzieleniem całkowitoliczbowym, to znaczy, że w wyniku obcięta zostaje ewentualna część ułamkowa.

```
19  5  / . 3 ok
20  5  / . 4 ok
```

Słowo MOD oblicza resztę z dzielenia całkowitoliczbowego.

```
19  5  MOD  .  4 ok
```

MOD pozwala w prosty sposób ustalić podzielność liczb.

```
154  7  MOD  .  0 ok
```

Jest także słowo /MOD pozwalające ustalić zarówno iloraz, jak i resztę z dzielenia całkowitoliczbowego,

```
19  5  /MOD .  .  3  4 ok
```

/MOD można łatwo zdefiniować z pomocą poznanych dotychczas słów.

```
: /MOD  2DUP  MOD  ROT  ROT  /  ;
```

W rzeczywistości w wielu implementacjach właśnie dzielenie definiowane jest z pomocą /MOD.

```
: /  /MOD  SWAP  DROP ;
```

Istnieją również użyteczne słowa do równoczesnego wykonania mnożenia dwóch liczb i ich dzielenia przez trzecią.

```
*/ ( n1 n2 n3 --- n1*n2/n3 )
*/MOD  ( n1 n2 n3 --- reszta wynik )
```

Dla działania tych operatorów ważne jest, że wynik pośredniego mnożenia może przekroczyć górną granicę liczb pojedynczej długości ze znakiem. Pozwala to, na przykład, z większą dokładnością obliczać procenty. Obliczmy obu metodami, ile wynosi 45% liczby 22220.

```
22220 45 100 */ . 9999 ok.
22220 100 / 45*. 9990 ok
```

Pierwszy wynik jest dokładniejszy, a ściślej biorąc całkiem dokładny. Łatwo to sprawdzić:

```
22220 45 100 */MOD .  . 9999 0 ok
```

Iloraz znajduje się na szczycie stosu, a reszta pod nim. Tu równa jest zeru.
Dwa słowa operują na pojedynczych liczbach. ABS daje jej wartość bezwzględną, MINUS (w innych implementacjach NEGATE) zmienia znak liczby.

I jeszcze dwa użyteczne słowa. MIN z dwóch liczb u góry stosu pozostawia tylko mniejszą, MAX - większą.

Niektóre proste działania arytmetyczne są często predefiniowane w słowniku. Niech nas nie zdziwi, że przeglądając słownik z pomocą VLIST znajdziemy w nim liczby 0,1,2,3. Po prostu dzięki skompilowaniu tych liczb działania z ich udziałem wykonają się szybciej. To samo dotyczy predefiniowanych działań, z których najczęściej spotykamy następujące:

```
1+  1-  2+  2-  2*  2/
```

Jeżeli któregoś z nich brakuje, możemy je łatwo zdefiniować. Na przykład

```
:  2/  2  /  ;
```

To słowo będzie dzielić liczby pojedynczej długości przez 2.

Istnieje także słowo +- (n1 n2 --- n3). Nadaje ono liczbie n1 znak n2 i zostawia na stosie jako n3.

3.6.2 Liczby podwójnej długości

W przypadku dodawania i odejmowania liczb podwójnych stosuje się odrębne operatory.

```
D+ /d1 d2 --- d1+d2/
```

```
D- /d1 d2 --- d1-d2/
```

Jeżeli tego drugiego operatora brakuje, to trzeba zmienić znak odjemnika z pomocą słowa DMONUS czy też DNEGATE . Poza tym dla liczb podwójnych istnieją podobnie działające odpowiedniki poznanych już słów, a mianowicie DABS, DMIN, DMAX i D+- .

Ostatnie wreszcie słowo użyteczne w działaniach arytmetycznych z udziałem liczb podwójnych to

```
S->D   n --- d
```

Przekształca ono liczbę pojedynczej długości w podwójną. W praktyce dokonuje się to prosto: nad liczbę znajdującą się na szczycie stosu wprowadza się 0 jako liczbę pojedynczej długości. Razem powstaje właściwa liczba podwójna.

3.6.3 Operacja mieszane

Czasem wygodnie jest wykonywać działania, w których uczestniczą liczby zarówno podwójnej jak i pojedynczej długości. Umożliwiają to następujące słowa

```
M/      ( d n1   --- n2=iloraz )
M/MOD   ( d n1   --- n2=reszta d2=iloraz )
U/MOD   ( ud u   --- u=reszta   u=iloraz )
U*      ( ud u   --- u=reszta   u=iloraz )
M*      ( n1 n2  --- d=iloczyn  )
U*      ( u1 u2  --- ud=iloczyn )
M*      ( d1 n u --- d2=wynik   )
```

Działanie tych słów wyjaśniają opisy w nawiasach. Ciekawe , że M*/ wytwarza "po drodze" wynik mnożenia w postaci liczby potrójnej długości.
Słownik nie zawsze zawiera wszystkie z tych słów.

Ćwiczenie 7

Spróbujmy obliczyć odręcznie, a potem sprawdźmy na komputerze wyniki następujących działań:

a/ 80 7 /MOD *
b/ 10 9 8 */MOD +
c/ 150 4 /MOD 4
d/ 6 7 5 MAX MIN

Ćwiczenie 8

W jaki sposób należy wprowadzić na stos dane, wykonać działanie i wyprowadzić wynik w następujących przypadkach?

a/ pomnożyć 32000 przez 29000
b/ pomnożyć 32000 przez 33000
c/ podzielić miliard przez 25 tysięcy
d/ podzielić z resztą 1500000335 przez 25000
e/ podnieść 777 do kwadratu
f/ podnieść 45333 do kwadratu.

3.7 Jak zmienić układ liczbowy?

W życiu codziennym stosujemy prawie wyłącznie jeden pozycyjny układ liczbowy: dziesiątkowy. Rzadko myślimy, że istnieją również inne układy. Czy dostrzegamy, na przykład, w jakim układzie mierzone są sekundy, minuty i godziny? Zdziwimy się, gdy ktoś powie, że strzałki na zegarku pracują w układzie sześćdziesiątkowym. Nawiasem mówiąc, właśnie ten układ był historycznie pierwszym pozycyjnym układem liczbowym. Wynaleźli go około 5 wieku przed naszą erą astronomowie chaldejscy. Oni też byli wynalazcami pozycyjnego zera, jakkolwiek dopiero w średniowieczu Hindusi i Arabowie docenili, jak ogromną rolę może ono spełniać.

Istotnym walorem FORTH-a jest to, że pozwala nam wyjść poza układ dziesiątkowy i wykorzystywać dodatnie cechy innych

układów. FORTH umożliwia wprowadzanie i wyprowadzanie liczb w dowolnym układzie liczbowym.

W dziesiątkowym na każdej pozycji jedynka jest 10 razy więcej warta niż jej sąsiadka z prawej. 10 nazywamy podstawą układu. Podobnie jest w innych pozycyjnych układach liczbowych, tylko wartość podstawy się zmienia.

Rzeczywiste działanie komputera najlepiej odzwierciedla najprostszy z układów pozycyjnych: dwójkowy. Są w nim tylko dwie cyfry; zera i jedynki. Inny układ niezmiernie użyteczny w pracy z komputerem - to omówiony już układ szesnastkowy.

FORTH ma specjalne słowo HEX, które pozwala natychmiast drukować liczby w układzie szesnastkowym i słowo DECIMAL, pozwalające powrócić do układu dziesiątkowego. A jak przechodzi się na inne układy? Służy do tego zmienna BASE przechowująca aktualną podstawę układu liczbowego stosowanego w przedstawianiu liczb. Wystarczy zmienić wartość BASE, by przejść w inny układ.

Jak dowiedzieć się, co w układzie hex reprezentuje liczba 255, największa liczba mieszcząca się w jednym bajcie? Napiszmy:

```
255 HEX . DECIMAL FF ok
```

A maksymalna liczba dwubajtowa? A maksymalna ze znakiem ?

```
65535 HEX U. DECIMAL FFFF ok
32767 HEX . DECIMAL 7FFF ok
```

By posługiwać się zmienną BASE musimy wyprzedzić o kilka stron wyjaśnienie sprawy zmiennych i zastosować nowe słowo: ! Wykrzyknik wprowadza pod adres, znajdujący się na szczycie stosu, w tym wypadku adres zmiennej, wartość z drugiej komórki stosu. Sprawdźmy działanie na paru przykładach:

```
255 2 BASE ! . DECIMAL 11111111 ok
32767 2 BASB ! CR . DECIMAL
111111111111111 ok
```

Wydrukowanych zostało piętnaście jedynek. Ostatni, szesnasty od prawej bit liczby dwubajtowej jest bitem znaku i gdy

przybierze wartość 1, w arytmetyce ze znakiem będzie to odczytane jako informacja, że liczba jest ujemna. Napiszmy linię, która liczbę 32768, a więc o 1 większą, wydrukuje jako liczbę bez znaku i liczbę ze znakiem. Zwróćmy uwagę, że w każdym z dotychczasowych przykładów piszemy na końcu DECIMAL i tym samym wracamy do układu dziesiątkowego.

```
32768 DUP 2 BASE ! CR U. CR . DECIMAL
1000000000000000
-100000000000000 ok
```

I wreszcie

```
32769 DUP 2 BASE ! CR U. CR . DECIMAL
1000000000000001
-111111111111111 ok
```

W ten sposób, przy pomocy liczb w układzie dwójkowym (binarnych) , namacalnie poznać możemy zasadę przedstawiania liczb ujemnych w formie tzw. uzupełnienia do dwóch.
I jeszcze jeden eksperyment ukazujący znaczenie układu szesnastkowego w pracy z komputerem. Przedstawmy dowolną liczbę w układzie dwójkowym i w hex.

```
27777 DUP 2 BASE ! CR . HEX CR . DECIMAL
110110010000001
6C81 ok
```

Podzielmy liczbę binarną na odcinki 4-bitowe poczynające od prawej strony i pod każdą czwórką bitów napiszmy cyfrę hex.

```
0110  1100  1000 0001
  6     C     8    1
```

Przekonujemy się, że każdy odcinek 4-bitowy ma taką samą wartość, jak cyfra w układzie szesnastkowym (C odpowiada dziesiętnej liczbie 12).

Ta właśnie odpowiedniość decyduje o użyteczności układu szesnastkowego w pracy z komputerem. Tylko w układzie

szesnastkowym i binarnym, trudnym do stosowania, możemy wyraźnie zobaczyć, że liczba złożona z dwóch bajtów umieszczonych w pamięci w kolejności odwrotnej zbudowana jest rzeczywiście z odpowiednich par cyfr szesnastkowych. W układzie dziesiątkowym jest to zupełnie niewidoczne. Ta i wiele innych przyczyn sprawiają, że zawodowi programiści z reguły szeroko korzystają w pracy z układu szesnastkowego. Warto zdobywać umiejętność posługiwania się nim.

Poznając FORTH-a starajmy się wykorzystać jego możliwości przedstawiania liczb w różnych układach numerycznych.

Zdefiniujemy teraz dwa użyteczne słowa. Pierwsze , H. , podaje wartość liczby w układzie szesnastkowym, ale nie zmienia bieżąco stosowanej podstawy. Drugie słowo, B?, pozwala ustalić, jakim układem posługujemy się w danej chwili. Nie możemy tej informacji odczytać ze zmiennej BASE, ponieważ w każdym układzie liczbowym znajduje się tam wartość 10. Dlaczego? Zastanówmy się sami.

Do obu definicji wprowadzamy nieznane dotychczas słowo @. Jest ono jakby odwrotnością wykrzyknika: pobiera liczbę dwubajtową spod podanego adresu, w tym wypadku adresu zmiennej BASE, i umieszcza na stosie.

```
: H.  BASE @ HEX OVER U. BASE ! ;

: B? BASE @ DUP DECIMAL .  BASE  ! ;
```

3.8 Liczby w pamięci i na stosie

Zdecydowana większość implementacji FORTH-a dokonana jest na komputery o tzw. adresowaniu bajtowym. Oznacza to, że liczba 16-bitowa lokowana jest w dwóch sąsiednich bajtach. Standard FORTH-a nie określa, jaka ma być kolejność tych bajtów w pamięci komputera. Autorzy interpretatorów z reguły kierują się konwencją stosowaną w danym komputerze.

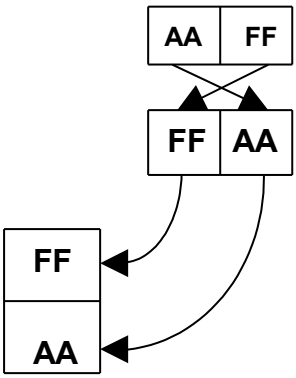
Tak na przykład, mikroprocesory, firmy Motorola umieszczają najpierw bardziej znaczący, a następnie mniej znaczący bajt liczby. Natomiast w mikroprocesorach Intela i Ziloga, na których oparta jest większość mikrokomputerów 8-bitowych używanych w Polsce, stosuje się konwencję odwrotną. W pamięci Atari, Commodore, Spectrum i innych komputerów liczba 16-bitowa przedstawiana jest zatem w odwrotnej kolejności

bajtów. Taki sam układ bajtów stosowany jest również w 16-bitowych mikroprocesorach Intela.

Dla użytkownika nie ma to bezpośrednio znaczenia, gdyż wszystkie systemy wyprowadzania danych pracują tak, że na ekranie czy wydruku cyfry są w prawidłowej kolejności. Ponieważ jednak FORTH jest językiem szczególnie ściśle powiązanym z wewnętrzną organizacją komputera, warto jest wiedzieć, w jakiej postaci liczby lokowane są przezeń w pamięci i na stosie. Ułatwia to dostęp nie tylko do liczb, lecz także do poszczególnych ich części składowych.

Losy liczby najłatwiej jest zrozumieć gdy przedstawi się ją w układzie szesnastkowym czyli w hex. Powiedzmy, że jest to liczba dwubajtowa: AAFF czyli 43775 dec. Pierwszą jej część, AA, stanowi bajt, który nazywamy bardziej znaczącym, górnym lub starszym. Po angielsku określa się go skrótem MSB od nazwy Most Signficant Byte.

FF - to druga, mniej znacząca część liczby, mieszcząca się w bajcie dolnym lub młodszym. Po angielsku oznacza się go skrótem LSB od nazwy Least Significant Byte. Dodajmy, że określenia MSB i LSB używane są również dla najstarszego i najmłodszego bitu w bajcie.



Rys. 1 Liczba pojedynczej długości kolejno: napisana w pamięci i na stosie

Rysunek 1 przedstawia kolejne przemiany, jakim ulega nasza liczba. Gdy ją umieścimy w pamięci komputera, na przykład z pomocą komendy

```
AAFF 9000 !
```

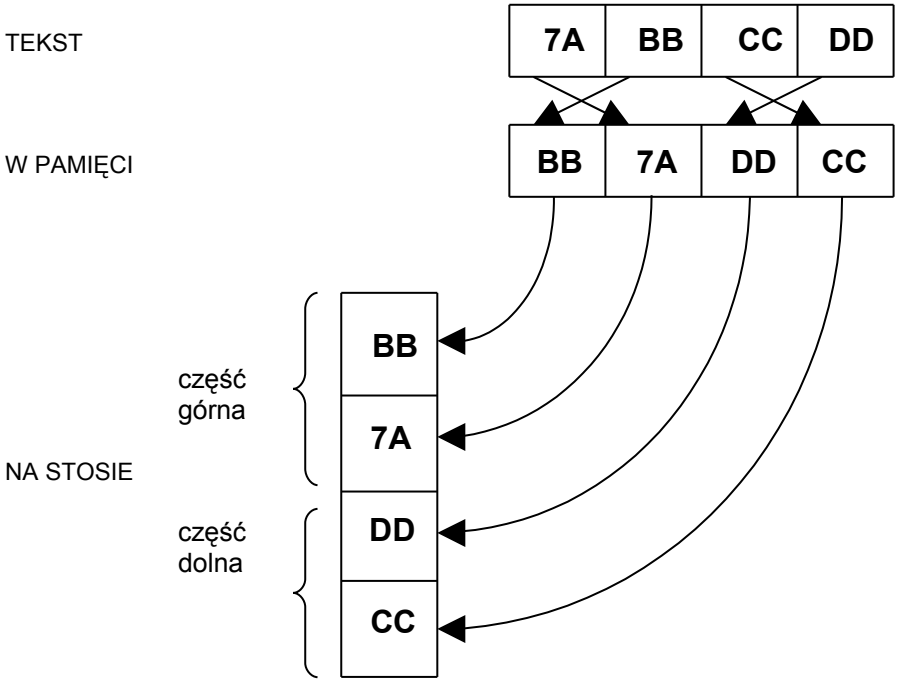
a następnie wyświetlimy zawartość dwóch komórek pod adresem 9000 hex, czyli 36864 dec, przekonamy się, że nasza liczba ma zamienione kolejnością MSB i LSB. Młodszy bajt jest z przodu. W BASIC-u nieraz do odczytania Liczby dwubajtowej stosujemy formułę: ?

```
? PEEK(128)+256*PEEK(129).
```

W hex trzeba stosować podobne przeliczenie, tylko mnożyć przez 100, co odpowiada liczbie 256 dec.

FORTH uwalnia od tych kłopotów, ponieważ ma zestaw słów automatycznie odczytujących wartość liczby.

Jeżeli wprowadzimy liczbę AAFF na stos i spróbujemy odczytać zawartość kolejnych bajtów stosu, to okaże się, że bliżej szczytu jest młodszy bajt, a pod nim starszy. Układ ten zgodny jest z kolejnością bajtów liczby w pamięci. Wszystkie stosy rosną ku dołowi i kolejność bajtów w pamięci i na stosie jest taka sama.



Rys. 2 Przemiany liczby podwójnej długości

Przyjrzyjmy się teraz w podobny sposób liczbie czterobajtowej czyli liczbie podwójnej długości. Rysunek 2 przedstawia przemiany liczby 7ABBCCDD. Zastrzeżmy ponownie, że to, co widzimy na rysunku, nie jest Rozwiązaniem jedynym, jakie się

stosuje. Otóż w tym wypadku starsza, bardziej znacząca część liczby podwójnej zachowuje w pamięci to samo położenie, co w druku, czyli "idzie przodem". Zamianie ulega tylko kolejność składających się na nią bajtów.

Wprowadźmy teraz naszą liczbę na stos:

```
7ABBCCDD. ok
```

Pamiętajmy o kropce, która sygnalizuje FORTH-owi, że jest to liczba podwójna. Jeżeli teraz napiszemy D. , zobaczymy liczbę taką, jak ją wprowadziliśmy, tyle że bez kropki. Spróbujmy jednak napisać:

```
7ABBCCDD.  U. U.  7ABB CCDD ok
```

Jak widać, na górze stosu znajduje się starsza część liczby podwójnej, którą wyprowadziło pierwsze słowo U. Znowu kolejność bajtów na stosie jest taka sama jak w pamięci, natomiast dwie części liczby podwójnej, inaczej niż przy liczbach pojedynczych, nie są zamienione miejscami.

Gdy nabierzemy pewnej wprawy w programowaniu w FORTH, przekonamy się, że ta pozorna łamigłówka ma istotne znaczenie praktyczne i rozszerza zakres możliwości operowania liczbami podwójnej długości.

ROZDZIAŁ 4
SŁOWA

4.1 Definiowanie

Jak powstają nowe słowa FORTH-a? Jeden sposób tworzenia słów już poznaliśmy. Jest nim definicja dwukropkowa. Jest ona podstawową i najczęstszą formą definiowania słów, ale nie jedyną. Spośród innych omówię trzy.

Pierwsza - to tworzenie s t a ł y c h z pomocą słowa CONSTANT. Wykonuje się to w następujący sposób:

```
12 CONSTANT TUZIN
```

Słowo TUZIN stało się równoważnikiem liczby 12. Pisząc TUZIN powodujemy, że 12 wchodzi na stos. Sprawdźmy.

```
TUZIN .S 12 ok
```

Każdej liczbie możemy w ten sposób przypisać nazwę. Słowo CONSTANT posłużyło również do wprowadzenia do predefiniowanego słownika FORTH-a szeregu stałych, o których wspominaliśmy: 0, 1, 2, 3, C/L, B/BUF, B/SCR. Wymieńmy jeszcze trzy stałe obecne z reguły w podstawowym słowniku.

BL - to liczbowy odpowiednik spacji w kodzie ASCII czyli 32 dec, a 20 hex, nieraz użyteczny.

FIRST zostawia na stosie adres początku pierwszego, dolnego buforu bloku służącego do współpracy z nośnikiem zewnętrznym: stacją dyskietek lub magnetofonem. LIMIT - podaje pierwszy wolny adres nad górnym buforem. Słowa te pozwalają zorientować się, ile miejsca i gdzie rezerwuje dany system na bufory. Są to zazwyczaj dwa, czasem trzy, kilobajty, W szeregu systemów o innej konfiguracji (patrz rozdz. 6) FIRST i LIMIT są zmiennymi.

Z m i e n n e są nam potrzebne częściej niż stałe. Do ich definiowania służy słowo VARIABLE. Wymaga ono przypisania

zmiennej początkowej wartości. Niektóre systemy czynią to automatycznie przypisując wartość 0, w innych musimy sami ją wpisać :

```
0 VARIABES FF
```

4.2 Wprowadzanie i odczytywanie wartości zmiennej

Zewnętrznie z pomocą VARIABLE definiuje się podobnie jak z CONSTANT. Jednakże w przeciwieństwie do BASIC-u napisanie FF nie wywoła wartości zmiennej. Napiszmy

```
FF  .S
```

A przekonamy się, że na stosie pojawi się duża liczba. Dzieje się tak dlatego, że w przypadku zmiennej napisanie nazwy wprowadza na stos jej a d r e s, a nie wartość. Mając już na stosie adres FF napiszmy poznane słowo

```
@  .  0 ok
```

Oto @ , zwane w gwarze programistów embrionem, pobrało spod adresu zmiennej FF jej wartość, a kropka wydrukowała ją. Można to uczynić prościej z pomocą innego słowa: znaku zapytania.

```
FF  ?  0 ok
```

Powoduje ono wydrukowanie liczby dwubajtowej spod adresu. Jak jednak przypisać zmiennej FF wartość? Oczywiście z pomocą poznanego już słowa ! , wykrzyknika.

```
7  FF  ! ok

FF  ?  7 ok
```

Zmienna otrzymała nową wartość. Pamiętajmy o kolejności liczb przy posługiwaniu się wykrzyknikiem: najpierw podajemy wartość, a potem adres. W tym wypadku adres wprowadzony został na stos przez nazwę zmiennej FF. Odwrotna kolejność może mieć czasem opłakane skutki. Np. zamiast 10000 pod adres 1000 wstawimy 1000 pod adres 10000 niszcząc tym jakąś definicję, Łatwość dostępu do pamięci, jaka cechuje FORTH-a, ma swój koszt: wymaga dokładności. Spróbujmy jednak wykonać tę samą czynność w BASIC-u, zwłaszcza gdy nie ma w nim komendy
DPOKE. Trzeba będzie przeliczyć 5000 na zawartość dwóch bajtów i wstawiać liczby dwu POKE-ami, pamiętając jeszcze, że najpierw powinien iść bajt młodszy.

FORTH ma odpowiedniki trzech omówionych przed chwilą słów operujące na liczbach jednobajtowych: C@ , C? i C!. Są także D@ , D! i D? dla liczb podwójnych. Wszystkich dziewięć słów pomaga w prostym wprowadzaniu i odczytywaniu danych. Każdy komputer ma podane w swej mapie pamięci komórki sterujące. Np. w Atari POKE 82,0 rozszerza o dwa znaki lewy margines. W FORTH piszemy 0 82 C! - i po wszystkim. W Commodore wartość z komórek 201 i 202 określa pozycję kursora. Łatwo ją odczytać z pomocą : 201 ?

Również kontakt ze zmiennymi zdefiniowanymi z pomocą VARIABLE może się odbywać osobno dla każdej z dwóch komórek, jakie VARIABLE rezerwuje na liczbę.

Istnieje ponadto słowo ALLOT pozwalające zarezerwować w zmiennej miejsce na dowolną liczbę bajtów. Czynimy to następująco:

```
0  VARIABLE  BB  100  ALLOT
```

ALLOT musi być użyte bezpośrednio po zdefiniowaniu zmiennej, rezerwuje bowiem miejsce w chwili, gdy jest użyte. Ma ono nader prostą definicję:

```
:  ALLOT  DP  +!  ;
```

Zmiany wartości zmiennej o określoną wielkość można dokonywać z pomocą słowa +! . Np.

```
2  FF  +!
```

zwiększy wartość FF o 2. Stosując liczbę ujemną zmniejszamy wartość.

Kolejny istotny sposób definiowania nowych słów polega na wykorzystaniu słowa definiującego USER dostępnego w wielu implementacjach. Tworzy ono w wyodrębnionej strefie pamięci FORTH-a tak zwane zmienne użytkownika. Jest to szczególny sposób definiowania zmiennych stosowany zwłaszcza w systemach komputerowych obsługujących wielu użytkowników. Jeżeli napiszemy

```
160  USER  Z
```

będzie to oznaczać zupełnie coś innego, niż w przypadku

VARIABLE. Liczba 160 stanowić będzie odległość zmiennej od początku strefy zmiennych użytkownika. Ponieważ każda zmienna zajmuje dwa bajty, oznaczać to będzie pozycje 80 w obszarze takich zmiennych.

Definiowanie z pomocą USER wymaga ostrożności. W strefie zmiennych użytkownika znajduje ok. 20 ważnych zmiennych, którymi posługuje się interpretator i do których użytkownik może: niejednokrotnie wprowadzać wartości bezpośrednio lub za pośrednictwem innych słów, jakie wykorzystuje.

Np. BASE jest jedną ze zmiennych użytkownika. Należą do nich między innymi FENCE określająca granicę, poniżej której z pomocą FORGET nie można skasować definicji, SCR zawierająca numer wykorzystywanego w danej chwili ekranu i STATE przybierająca wartość niezerową gdy FORTH znajduje się w fazie kompilacji.

Wymieniliśmy cztery podstawowe sposoby definiowania słów: z pomocą definicji dwukropkowej oraz słów definiujących CONSTANT, VARIABLE i USER. Do innych sposobów wrócimy w drugiej części książki.

4.3 A gdzie to jest?

Przy dużej różnorodności form definiowania słów znamienne dla FORTH-a jest to, że wszystkie słowa mają taką samą podstawową strukturę i wszystkie powstają w tym samym miejscu: na szczycie słownika. Słownik FORTH-a jest ciągiem słów w szczególny sposób połączonych między sobą, co uniemożliwia "wciśnięcie" nowego słowa między dwa wcześniej zdefiniowane. Słowo może być dołączone do słownika tylko na Jego szczycie czyli tam, gdzie w danej chwili jest adres pierwszej wolnej nad nim komórki pamięci.

Z tego względu duże znaczenie mają słowa DP i HERE, które adres ten podają.

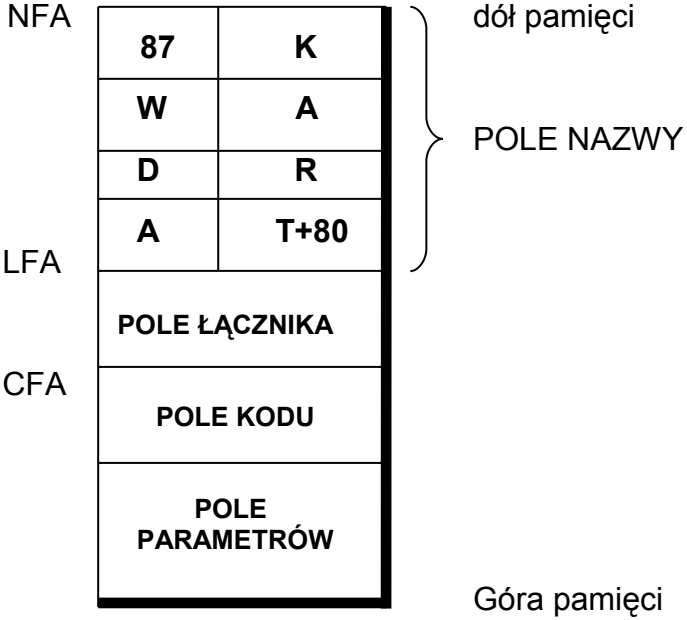
DP, dictionary pointer - wskaźnik słownika , to zmienna użytkownika przechowująca ów zmieniający się adres. HERE pozwala bezpośrednio odczytać wartość DP i jest zdefiniowane następująco:

```
:  HERE DP @ ;
```

Przystępując do definiowania czegokolwiek, dokonujemy tak zwanego wejścia słownikowego (ang. dictionary entry), ingerujemy w stan słownika. Czasem popełniamy błąd i na szczycie słownika powstaje wówczas nazwa, która nic nie znaczy. Dobrze jest przystępując do pracy w FORTH umieścić na szczycie słownika puste słowo, np. NOOP, by potem pisząc FORGET NOOP móc zaśmiecenie usunąć, oczywiście, przed zdefiniowaniem nowych słów.

4.4 Budowa słowa

Zewnętrznie biorąc słownik jest listą sekwencyjną słów, listą jakich wiele spotkać można w programowaniu. W rzeczywistości słownik FORTH-a jest strukturą bardzo oryginalną, przede wszystkim dlatego, że pulsuje stale życiem i niezwykle sprawnie wykonuje różnorodne funkcje. Wyjaśnienie tych cech tkwi w poważnym stopniu w strukturze pojedynczego słowa FORTH-a. Przyjrzyjmy się jej bliżej na przykładzie wcześniej poznanego słowa KWADRAT.



Rys. 3 Budowa słowa. Liczby w hex

Na rysunku 3 przedstawiony jest schemat budowy każdego słowa FORTH-a stosowany w większości implementacji. tradycyjnie przedstawia się go jakby odwrócony "do góry nogami", dołem pamięci ku górze, ułatwia to bowiem czytanie. W takiej kolejności pojawiają się zresztą elementy słowa również na ekranie.

Słowo składa się z czterech pól. Pierwsze z nich - to pole nazwy (ang. name field) . Składa się na nie otwierający bajt długości oraz po jednym bajcie na kod ASCII każdego znaku w nazwie. Kod ostatniego znaku automatycznie zwiększany jest o 80 hex czyli 128 dec. Pomaga to interpretatorowi w znajdowaniu słów w czasie poszukiwać w słowniku.

Warto zatrzymać się chwilę przy bajcie długości, ponieważ pełni on szereg funkcji, które będą omówione dalej. Przedstawmy wewnętrzną budowę bajtu. Ponumerujmy przy tym bity od prawej cyframi od O do 7. Cyfry te odpowiadają kolejnym potęgom liczby 2 reprezentującym wartości jedynki w poszczególnych bitach. Pod numerami bitów podajemy te wartości w układach liczbowych dec i hex.

Bity	7	6	5	4	3	2	1	0
dec	128	64	32	16	8	4	2	1
hex	80	40	20	10	8	4	2	1
	1	P	S	Liczba znaków nazwy				

Rys. 4 Budowa bajtu długości nazwy

Wyjaśnijmy na podstawie rysunku znaczenie poszczególnych bitów. S pięciu mniej znaczących podawana jest długość nazwy w bajtach. Proste obliczenie wskazuje, że maksymalna długość nazwy może wynosić 31 znaków. Wielkość tę zawiera zmienna WIITH, do której użytkownik może, jeżeli uzna to za celowe, wprowadzić mniejszą wartość. Nie stosuje się dziś na ogół, jak widzimy, surowych ograniczeń, które tak dokuczały Moore'owi przy wyborze nazwy nowego języka.

Należy podkreślić, że FORTH dopuszcza w nazwie słowa wszystkie znaki kodu ASCII, które dadzą się wydrukować. Część znaków pełni, jak wiemy, specjalne funkcje, np. strzałki do przemieszczania kursora, i takie znaki nie mogą

być zwykle drukowane. Nie ma ponadto stosowanego w wielu językach ograniczenia co do pierwszego znaku nazwy. Może być on dowolny. W pełni poprawna byłaby, powiedzmy, taka nazwa:

%/!+:=,-„

Zajmijmy się pozostałymi trzema bitami. Najbardziej znaczący bit z zasady ustawiony jest na 1, to znaczy bajt długości nazwy nigdy nie ma wartości mniejszej niż 128 dec, czyli 80 hex. To także ułatwia interpretatorowi jego znalezienie. Specjalne znaczenie mają pozostałe dwa bity oznaczone S i P.

Działanie bitu S (od angielskiego słowa smudge), któremu w tym wypadku można nadać przybliżone znaczenie: znamię, poznamy za kilka chwil. Bit P, mający numer 6, a siódmy od prawej - to bit pierwszeństwa (ang, precedence bit) . Jego znaczenie omówimy w drugiej części książki, w punkcie 8.2 poświęconym tzw. słowom natychmiastowym.

Przejdźmy do następnego pola z rysunku 3. Jest to pole łącznika (ang. link field) . W jego dwóch bajtach mieści się adres początku pola nazwy poprzedniego słowa. Łączniki kolejnych słów wiążą je między sobą i pełnią doniosłą rolę w przeszukiwaniu słownika przez interpretator.

Kolejne, trzecie pole - to pole kodu (ang. code field). Zawiera ono adres kodu wykorzystywanego w danym słowie. Gdy słowo jest w kodzie maszynowym, zazwyczaj znajduje, się on bezpośrednio pod polem kodu czyli dwa bajty dalej. Nie jest to jednak konieczne i kod maszynowy słowa można umieścić zupełnie gdzie indziej.

Natomiast szczególnie często adres w polu kodu rozmaitych słów jest taki sam. Gdy mianowicie słowo zostało zdefiniowane z pomocą dwukropka, CONSTANT czy VARIABLE, pole kodu wskazuje adres kodu maszynowego jednego z tych słów definiujących. Tak więc po zawartości pola kodu możemy rozpoznać, w jaki sposób słowo zostało zdefiniowane.

Trzy dotychczas omówione pola słowa noszą łącznie nazwę jego nagłówka (ang. header lub head), czwarte stanowi jego ciało (ang. body).

Jest to przeważnie, choć nie zawsze, największe pole parametrów (ang. parameter field).

Co zawiera? Dla laika, który po wyświetleniu fragmentu pamięci napotyka ciąg nieuporządkowanych liczb, jest zgoła niezrozumiałe. W rzeczywistości pole parametrów zawiera adresy pól kodów słów wchodzących w skład definicji, a także ewentualne parametry liczbowe i tekstowe. Budowa niektórych słów jest znacznie bardziej skomplikowana niż w omawianym przykładzie. Jednakże zawsze mają one cztery wymienione części składowe. W naszym przypadku słowa KWADRAT w polu parametrów znajdą się adresy słów: DUP * . ; .

Rysunek 5. zamieszczony na sąsiedniej stronie, przedstawia to poglądowo.

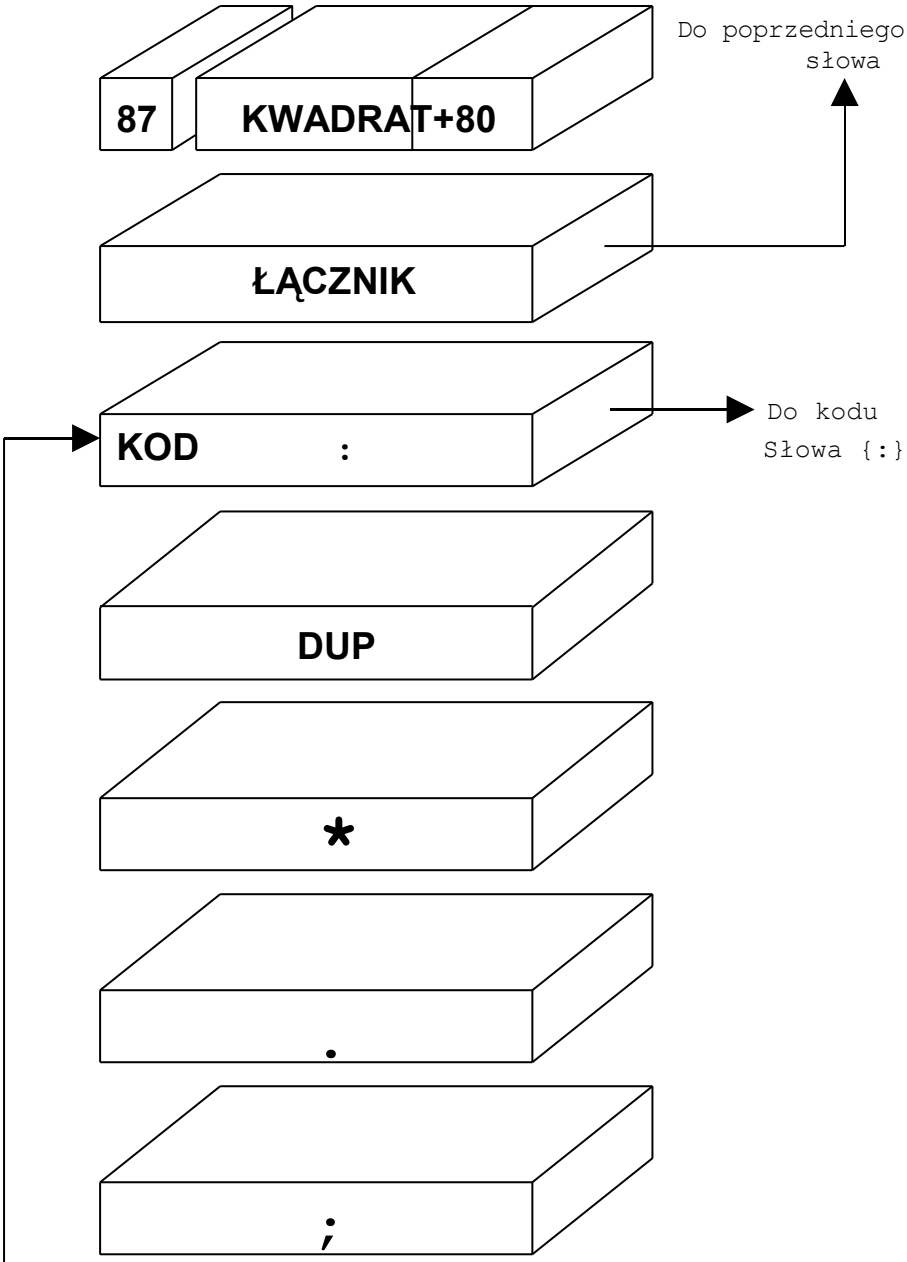
Co dzieje się ze słowem, gdy je definiujemy, oraz wtedy, gdy jest wykonywane? Podczas definiowania FORTH najpierw zakłada jakby pusty blankiet słowa. Wpisuje do niego w bajcie długości liczbę nazwy powiększoną o 128 czyli 80 hex. Potem w kodzie ASCII wpisuje kolejne litery nazwy. Wartość ostatniej litery zwiększa znowu o 128. W ten sposób początek i koniec pola nazwy są wyraźnie zaznaczone.

FORTH czyni jeszcze jedno: zwiększa o 32 wartość bajtu długości słowa, co oznacza, że ustawia na 1 bit nr 5 w tym bajcie. Jest to bit, który oznaczyliśmy na rysunku 4 literą S. FORTH zaznacza w ten sposób, że słowo, którego nagłówek został wprowadzony do słownika, nie jest jeszcze zdefiniowane.

Następują kolejne czynności. FORTH odczytuje, jakiego słowa definiującego użyliśmy i jego adres wstawia do pola kodu. Może to być np. adres kodu słowa : .

Potem, w miarę jak piszemy definicję, po każdym naciśnięciu klawisza Return interpretator poddaje analizie kolejną porcję wprowadzonych słów i ewentualnie liczb, a gdy weryfikacja wypada pomyślnie - adresy kolejnych słów oraz liczby i ewentualnie teksty w kodzie ASCII umieszcza w polu parametrów definiowanego słowa.

Nadchodzi wreszcie końcowy średnik. Gdy definicja jest poprawna, FORTH wpisuje do pola parametrów adres słowa ; i kończy tym samym definiowanie słowa.



Rys. 5 Słowo FORTH-a

Czyni ponadto jeszcze jedno: ustawia na 0 bit nr 5 bajtu długości słowa, bit S. Zaznacza w ten sposób, że słowo stało się ważne, że może być odtąd przez nas stosowane. W niektórych implementacjach czynności zmiany bitu S mają odwrotną kolejność. Najpierw otrzymuje on wartość 0, a dopiero po zdefiniowaniu słowa ustawiany jest na 1.

A co się stanie, gdy zechcemy niepoprawnie zdefiniować nowe słowo? Nastąpi znana nam reakcja w postaci sygnału błędu, a słowo pozostanie w słowniku FORTH-a niby niedokończona budowla, świadectwo naszego niepowodzenia. Pozostanie także inny wyraźny tego ślad: ustawiony na 1 bit nr 5 w pierwszym bajcie. Jako umieszczanie znamienia można określić to, co wykonuje FORTH i co my także możemy uczynić z pomocą słowa SMUDGE.

Nasz KWADRAT został pomyślnie zdefiniowany. Chcemy go wypróbować i piszemy:

```
7 KWADRAT
```

Co się teraz dzieje? FORTH przystępuje natychmiast do wykonania naszego słowa. Nie zawsze, oczywiście, jest ono na szczycie słownika, czasem po drodze zdefiniowaliśmy wiele słów. Wobec tego FORTH szuka słowa. Przeszukuje słownik od HERE korzystając z łączników, odczytuje jedną nazwę po drugiej, aż wreszcie ... jest! Jest nasz KWADRAT.

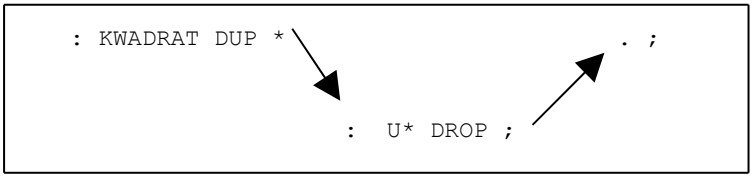
I oto następuje to, co jest efektem wynalazku Charlesa Moore'a, a czego w chwili narodzin FORTH-a nie znał żaden język programowania. FORTH rozpoczyna błyskawiczny bieg. Ma niby turysta wyraźne znaki szlaku, którym powinien podążać: wszystkie kolejne adresy potrzebne do wykonania słowa KWADRAT. Biegnie więc. Nie jest to jednak szlak turystyczny, lecz komputer! Bieg trwa ułamki sekund, mikrosekundy. I oto jest już wynik, jest sygnał: drukuj 49.

We wcześniejszym przygotowaniu kodu - szczególnego, bo poza nazwą złożonego z samych adresów - tkwi tajemnica szybkości FORTH-a. Wszystko jest gotowe, gdy się startuje.

Popełniliśmy drobne uproszczenie. Słowo * nie jest primitivem, lecz powstaje w wyniku definicji:

```
: * U* DROP ;
```

Gdy zatem wykonanie dotarło do znaku mnożenia, FORTH skoczył pod ten adres, a tam znalazł znów kolejno adresy dwukropka, U, DROP i średnika, na których podstawie wykonał kolejne słowa. Można to przedstawić następująco:



Rys. 6 Kod zszywany

Rysunek ten wyjaśnia z grubsza istotę kodu zszywanego. Jest on właśnie jakby zszywany z pomocą adresów kolejnych definicji. W bardziej skomplikowanych słowach rysunek taki rozrósłby się i stworzył wiele pięter. Ta pajęcza sieć połączeń zawdzięcza wielką sprawność przede wszystkim temu, że maszynowy skok pod adres trwa w komputerze bardzo krótko, że nie zależy od położenia komórek w pamięci, poza szczególnym przypadkiem komórek, na stronie zerowej, dostępnych często w jeszcze mniejszym ułamku sekundy.

Dlatego m. in. możliwość ulokowania stosu głównego FORTH-a na stronie zerowej, czyli poniżej 256 bajtu, jaką zapewniają mikroprocesory 6502, 6510 i pokrewne w Apple II, Atari, Commodore i wielu innych komputerach, sprzyja dodatkowo szybkości FORTH-a.

Ćwiczenie 9

Przedstawmy z pomocą rysunku podobnego do wykonanego powyżej kod zszywany słowa 2ROT. Weźmy pod uwagę, że 2ROT wykorzystuje 2SWAP oraz że ROT definiowane jest następująco:

```
: ROT  >R  SWAP  R>  SWAP  ;
```

4.5 Dostęp do słowa

Niejednokrotnie użyteczne staje się dotarcie do poszczególnych pól słowa, np. w celu wprowadzenia do zdefiniowanej stałej nowej wartości, co nie jest tak proste, jak w przypadku zmiennej. Służy do tego poniższy zestaw słów:

' cccc Z pomocą apostrofu, znaku znajdującego się na klawiaturze komputera zwykle nad cyfrą 7, wywołujemy adres pola parametrów słowa, którego nazwę umieścimy jako następną.

NFA / --- pfa nfa/

LFA / --- pda lfa/

CFA / --- pfa cfa/

PFA / --- nfa pfa/

Cztery ostatnie słowa w sposób wskazany w nawiasach zmieniają adres pola parametrów w inny, a w przypadku PFA adres pola nazwy w adres pola parametrów. W niektórych implementacjach występują pewne różnice w stosunku do przedstawionego tu trybu przekształcania adresów.

Gdy zdefiniowaliśmy słowo, często chcielibyśmy zobaczyć, jaki jest jego wygląd w pamięci. Można to osiągnąć z pomocą słowa DUMP, którego często nie ma w słowniku podstawowym, ale które może być łatwo zdefiniowane. Oto definicje dwóch wersji Pierwsza, prostsza, wyświetla samą tylko zawartość kolejnych bajtów, druga - również adres co ósmego bajtu.

```
a) : DUMP 0 DO DUP C@ . 1+ LOOP ;

b) : LDMP DUP 8 + SWAP DO I C@ 4 .R LOOP ;

   : DUMP2 OVER + SWAP DO CR I 5 .R I LDMP
          8 +LOOP CR ;
```

DUMP oczekuje na stosie adresu pierwszego bajtu i liczby bajtów.

Łatwo uzyskamy teraz obraz pamięci słowa zdefiniowanego jako ostatnie. LATEST wprowadza na stos adres pola nazwy tego

Słowa, HERE - adres pierwszej po nim wolnej komórki. Różnica tych wielkości - to długość definicji słowa.

HEX LATEST HERE OVER - DUMP DECIMAL

pozwoli obejrzeć to słowo w postaci liczb w hex. Pierwszy bajt powinien mieć wartość 80 + liczba liter w nazwie.

W przypadku słowa z innego miejsca w słowniku należy posłużyć się opisanymi wcześniej słowami. Oto możliwy sposób postępowania:

```
: ' <nazwa> ( pfa1 )
NFA         ( nfa1 )
'  <nazwa-następnego-słowa>     ( nfa1  pfa2 )
NFA         ( nfa1  nfa2 )
OVER        ( nfa1  nfa2  nfa1 )
-           ( nfa1  n )
DUMP  ;
```

ROZDZIAŁ 5
KONSTRUKCJE PROGRAMOWANIA

5.1 Czym jest programowanie strukturalne?

W omawianiu FORTH-a doszliśmy do kwestii szczególnie ważnej: opisu jego s t r u k t u r powodujących zmianę sekwencyjnego porządku wykonywania programu. To właśnie istnienie konstrukcji pozwalających budować w programie rozwidlenia, na których wybór drogi zależy od spełnienia bądź niespełnienia określonego warunku, powtarzać ciągi działań określoną liczbę razy albo aż do spełnienia warunku - w poważnej mierze określa olbrzymie możliwości komputera nie tylko w wykonywaniu obliczeń, lecz zwłaszcza w rozwiązywaniu p r o b l e m ó w.

FORTH rozporządza przemyślanym i użytecznym zestawem takich konstrukcji. Może być zaliczony do języków, które pobudzają i umożliwiają programowanie strukturalne. Ten typ programowania jest przedmiotem wielu dociekań i dyskusji, które w roku 1972 zainicjowała głośna książka Dahla, Dijkstry i Hoare'a "Structured programming". Głównym twórcą tej koncepcji był Edsger W. Dijkstra - jeden z najwybitniejszych informatyków współczesnych.

We wspomnianej książce uznano dość rygorystycznie za dopuszczalne stosowanie jedynie trzech podstawowych operatorów: IF...THEH...ELSE, IF...THEN i DO...WHILE. Późniejsza dyskusja doprowadziła do nieznacznego rozszerzenia tego katalogu, a także do uznania, że skok GOTO w pewnych warunkach może być w ograniczonym zakresie stosowany. Omawiając FORTH warto w tym miejscu podkreślić: rozporządza on w s z y s t k i m i podstawowymi konstrukcjami programowania strukturalnego.

Ten typ programowania nie jest jedynie ciekawym pomysłem teoretyków informatyki. Jest przede wszystkim potrzebą i

wymogiem praktyki. Joseph M. Fox, który przez lata kierował wydziałem programowania firmy IBM, opisuje znamienny przykład [17]. W r. 1969 IBM wygrywając przetarg zawarła kontrakt na opracowanie za milion dolarów systemu automatyzacji przechowywania materiałów informacyjnych dla dziennika "New York Times". Ponowne sprawdzenie wykazało, że firma nie zmieści się w umownej cenie i że grozi jej strata 800 tys, dolarów. Kierownictwo IBM powierzyło wówczas zorganizowanie prac zatrudnionemu od roku H. Millsowi, który miał już za sobą karierę wybitnego matematyka i administratora.

Stworzona przez niego grupa pracując metodami programowania strukturalnego w ciągu 22 miesięcy opracowała system, w którym posłużono się 83 tysiącami operatorów w języku wysokiego poziomu. IBM zyskało 300 tysięcy dolarów.

Efektem owych doświadczeń było to, że zgodnie z wcześniejszymi naleganiami Fox'a kierownictwo IBM zdecydowało się na wydanie sporych sum, by w ciągu 2 tygodni przeszkolić w programowaniu strukturalnym 2600 osób. Wyniosło to, jak zauważa autor książki, 100 człowieko-lat, lecz było słuszne.

5.2 Porównania

Zasadniczą podstawą działania wszystkich struktur jest sprawdzanie, czy określony warunek jest spełniony, czy też nie. "Jeżeli A równa się B, to rób to a to, a jeżeli nie równa się, rób tamto". "Powtarzaj czynność dopóki wskaźnik pętli nie zrówna się z granicą". I tak dalej.

W tych i wielu innych przypadkach wymaga to dokonywania porównań. FORTH rozporządza grupą słów pozwalających wykonać takie porównania. Ogólnie biorąc ich wynikiem jest znacznik logiczny "prawda" lub "fałsz". Znacznik taki oznaczać będziemy literą f od angielskiego słowa flag i rozróżniać obie postacie znacznika: tf - prawda, ff - fałsz.

Podobnie jak w wielu innych językach wysokiego poziomu, znacznik fizycznie nie różni się od liczb. Specyfika FORTH-a polega na tym, że wszystkich porównań dokonuje się na wartościach, które znajdują się na stosie. Porównywane parametry muszą być zatem wprowadzone na stos, a w wyniku porównania znikają i zostają zastąpione przez znacznik logiczny f. Jest on zawsze liczbą pojedynczej długości, przy czym zero oznacza fałsz, a każda wartość niezerowa - prawdę.

5.2.1 Porównywanie liczb pojedynczej długości

Trzy podstawowe operatory porównań (=,<,>) nie różnią się działaniem, poza stosowaniem ONP, od podobnych w innych językach.

Przykłady:

```
7  VARIABLE  AA ok
AA  @   .  (Return)  7 ok
21  28  >  .   0 ok
21  28  <  .   1 ok
```

Do porównania liczb bez znaku służy słowo:

```
U<  (u1 u2 --- f)
```

Trzeba je stosować wtedy, gdy porównywane liczby są większe niż 32767. Na przykład:

```
40000 41000 <  . 0 ok
40000 41000 U<  . 1 ok
```

Kolejne operatory działają na pojedynczych liczbach i sprawdzają ich relację w stosunku do zera. Szczególnie często stosowane jest 0=, które jest odpowiednikiem logicznego ,NOT.

```
0=  ( n --- f )
0>  ( n --- f )
0<  ( n --- f )
```

Wszystkie te podstawowe słowa będące operatorami relacji umożliwiają tworzenie dalszych. Należy pamiętać, że słowa < , =, > działają zawsze na parze górnych liczb na stosie, i zastępują ją znacznikiem, nie można więc użyć dwóch takich operatorów naraz. Na przykład

```
A  B  <  =
```

wywoła błąd, ponieważ dla drugiego porównania zabraknie liczby, łatwo jest jednak zdefiniować słowa odpowiadające analogicznym istniejącym np. w BASIC-u.

```
: <=  >  0=  ;
: =>  <  0=  ;
```

Słowo 0= jest w FORTH bardzo przydatne: pozwala zamienić znacznik logiczny "prawda" lub "fałsz" na odwrotny. Jest to często potrzebne, ponieważ, jak zobaczymy za chwilę, znacznik logiczny znajdujący się na stosie głównym steruje działaniem rozwidleń, odgałęzień i pętli warunkowych. Często zdarza się, że potrzebny jest nam do sterowania programem akurat przeciwny znacznik.

Ćwiczenie 10

Przyjmując, że nie mamy słowa 0> , tak często bywa, zdefiniujmy je przy pomocy 0< .

5.2.2 Porównywanie liczb podwójnej długości

Do porównywania liczb podwójnej długości stosować trzeba inne operatory.

```
D= /d1 d2 --- f/
D< /d1 d2 --- f/
```

Nie zawsze występują one i wtedy trzeba je samemu zdefiniować. Można to uczynić ustaliwszy położenie górnych i dolnych części liczb podwójnych na stosie.

Zdefiniujemy słowo, które będzie sprawdzać, czy liczba mieści się w zadanym przedziale.

```
: PRZEDZIAL  ( dolna-granica n gorna-granica )
       OVER  ( dol-gr n gor-gr n )
          >  ( dol-gr n f1 )
         >R  ( dol-gr n >
          <  ( f2 )
          R> ( f2 f1 )
*  ;        ( f )
```

Sprawdźmy:

20 370 400 PRZEDZIAŁ . 1 ok

Mnożenie obu znaczników pozwoliło ustalić, czy żaden z nich nie równa się zeru, co świadczyłoby o niespełnieniu jednego z warunków.

5.3 Jeżeli... to

Para lub trójka słów bada w FORTH spełnienie warunku i zależnie do wyniku sprawdzenia obiera dalszą drogę programu. Są to konstrukcje złożone ze słów

IF ... THEN

lub

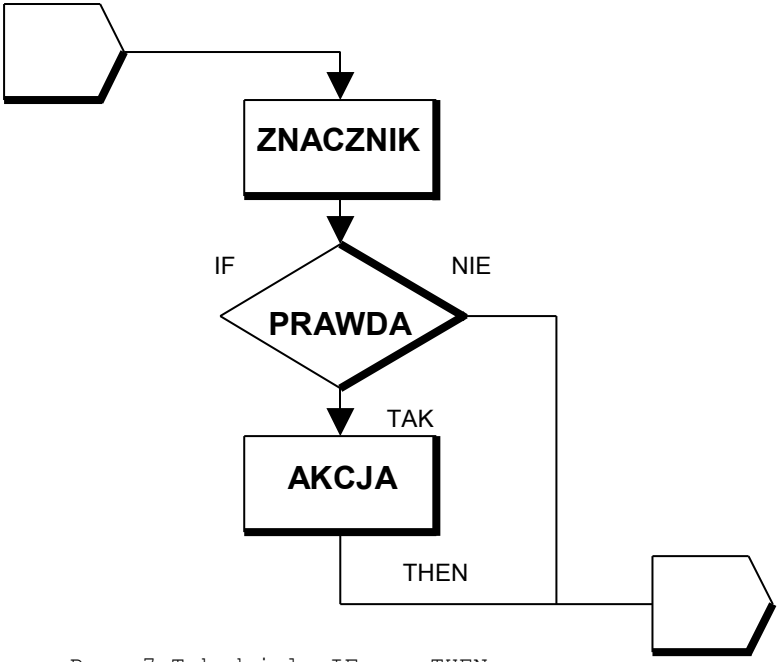
IF ... ELSE ... THEN

Jak działają? Należy raz jeszcze wskazać na ważną cechę FORTH-a. Wszystkie struktury wykonujące rozwidlenia na podstawie warunku dokonują jego sprawdzenia na stosie głównym. W danym wypadku takim kontrolerem spełnienia warunku jest słowo IF. Czyni to w prosty sposób: sprawdza, czy na szczycie stosu znajduje się wartość niezerowa, czyli znacznik "prawda" i w takim wypadku uruchamia się ciąg działań określonych za IF Jeżeli natomiast na stosie jest zero, czyli znacznik "fałsz", to ciąg ten nie jest wykonywany, a realizacja przenosi się za THEN w pierwszej konstrukcji, a za ELSE w drugiej.

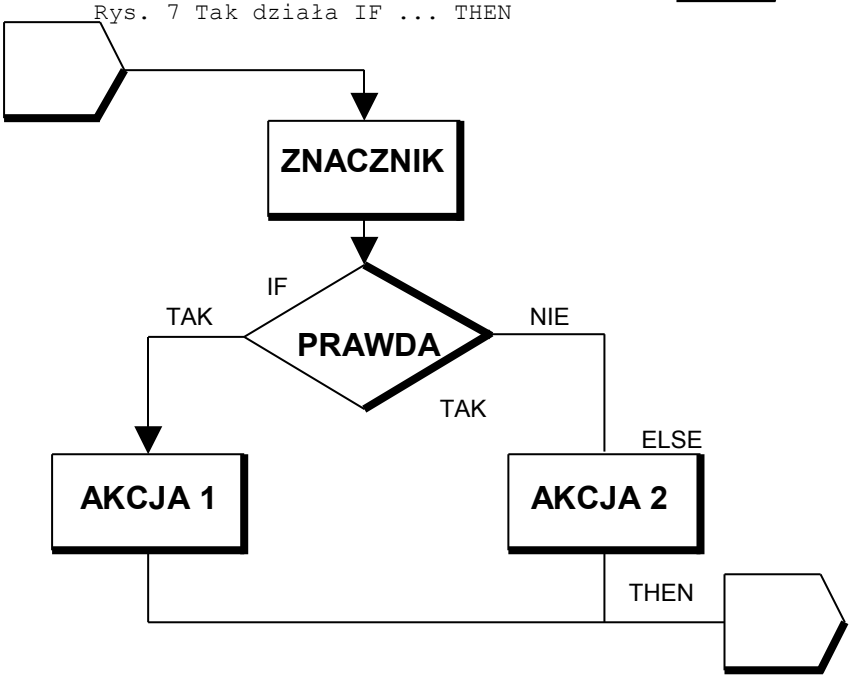
Ważna uwaga: IF zdejmuje ze stosu sprawdzaną wartość.

Słowo THEN zastępowane jest nieraz przez ENDIF. Zazwyczaj umieszcza się w słowniku oba, przy czym w fig-FORTH pierwotne jest na ogół ENDIF, a THEN zdefiniowane jest z jego pomocą, czyli jest o mało znaczący ułamek sekundy wolniejsze. Sposób, w jaki zdefiniowane są IF, ELSE i ENDIF, Czytelnik pozna w drugiej części książki, w rozdziale 9. Na tym miejscu poczyńmy dwie uwagi natury praktycznej..

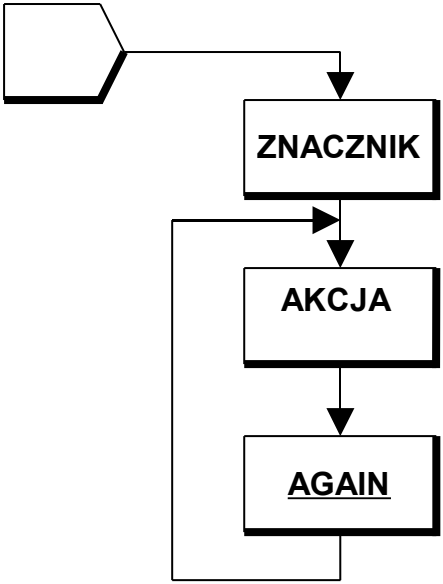
Po pierwsze, IF, THEN, ELSE i wszystkie pozostałe słowa omówione w tym rozdziale działają tylko wewnątrz definicji i są tzw. słowami natychmiastowymi.



Rys. 7 Tak działa IF ... THEN



Rys. 8 Tak działa IF ... ELSE ...THEN



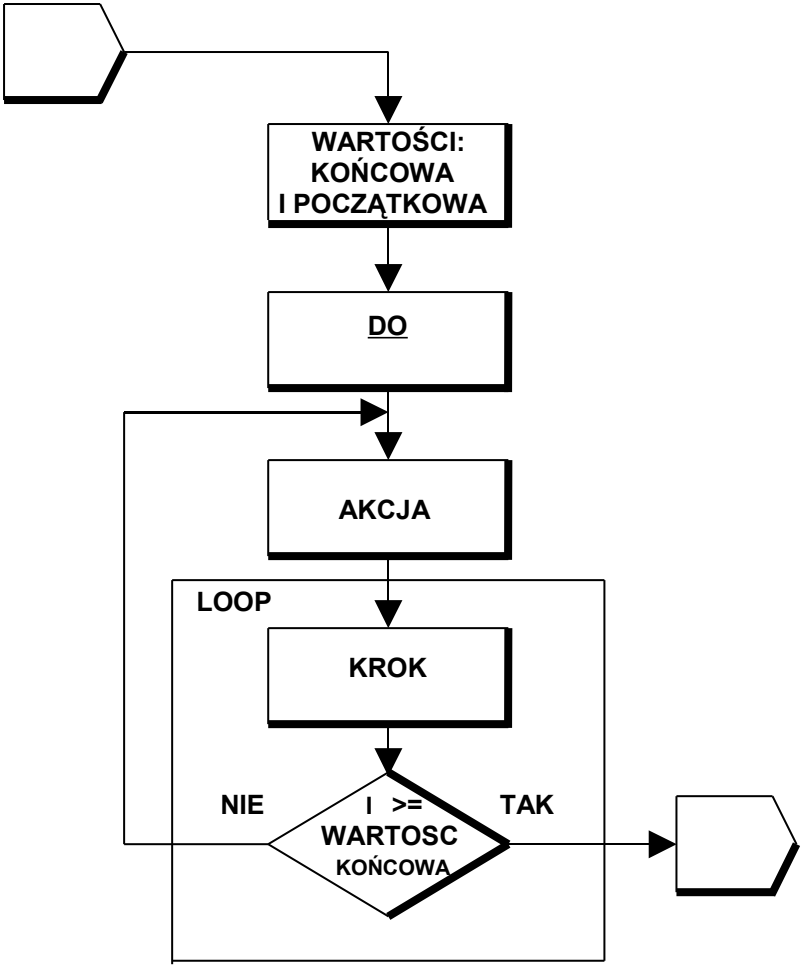
Rys. 9 Pętla podstawowa BEGIN...AGAIN

Po drugie, musi ich być zawsze jeden komplet dla każdej czynności. Nie można dać dwóch IF i jednego THEN a nie odwrotnie. Interpretator sprawdza to i sygnalizuje ewentualny błąd.

Do wyjaśnienia pozostała rola ELSE. Otóż gdy IF napotyka na stosie 0, traktuje to jako niespełnienie warunku i akcja przenosi się tuż za ELSE, a po wykonaniu ciągu określonych tam działań - za THEN. Taki wybór jednej z dwóch dróg jest często użyteczny Działanie obu struktur przedstawiamy na schematach blokowych na poprzedniej stronie. Jest to forma ukazywania i analizy programów i ich fragmentów bardzo pomocna w fazie projektowania i uruchamiania programów.

5.4 Pętla podstawowa

W FORTH, podobnie Jak w Pascalu, dostępne są pętle czyli struktury powodujące powtórzenie czynności, sterowane przez sprawdzenie warunku albo z pomocą wskaźnika, czyli przez obliczanie. Istnieje również pętla, w której nie jest przewidziane jej zakończenie, swoista taśma bez końca. W naszych programach



Rys.10 Schemat blokowy DO...LOOP

Jest ona niezbyt użyteczna, natomiast warto wiedzieć, że taką właśnie pętlę stanowi w FORTH np. mechanizm powodujący, że komputer stale czeka na to, co wprowadzimy z klawiatury. W FORTH pętlę podstawową wyodrębniają słona

BEGIN...AGAIN

Przedstawia ją rysunek 9 na poprzedniej stronie.

5.5 Pętle liczone: DO...LOOP i DO...+LOOP

Pętlę, która działa na zasadzie odliczania kolejnych powtórzeń, poznaliśmy już w rozdziale wstępnym. Jest to konstrukcja:

DO ... LOOP

Warto przyjrzeć się bliżej działaniu tej pętli posługując się rysunkiem 10. Jak pamiętamy, wymaga ona wprowadzenia na stos dwóch liczb: górnej zwiększonej o 1 i dolnej granicy pętli. Słowo DO umieszcza obie te liczby na stosie powrotów kasując je na stosie głównym i inicjuje powtarzanie czynności określonych w następnych słowach. LOOP porównuje obie wartości i jeżeli najwyższa, oznaczająca bieżący wskaźnik, jest mniejsza od wartości granicznej, to zwiększa wskaźnik o 1 i przekazuje wykonanie wstecz do słowa znajdującego się bezpośrednio za DO. W przeciwnym wypadku obie liczby na stosie powrotów zostają skasowane i wykonanie przechodzi do pierwszego słowa za LOOP, czyli pętla kończy się. Mechanizm ten wyjaśnia także działanie niezmiernie użytecznego słowa I stanowiącego wskaźnik pętli. Po prostu I odczytuje wskaźnik pętli znajdujący się w danej chwili na szczycie stosu. Ma zatem identyczne działanie, jak wcześniej omówione R i jest przez nie definiowane. Konstrukcje DO...LOOP mogą być zagnieżdżane "piętrowo".

Przykłady:

Zilustrujmy działanie DO...LOOP definicjami słów generujących liczby ciągu Fibonacciego. Ciąg ten, wynaleziony w 12 wieku przez jednego z największych uczonych średniowiecza Leonarda z Pizy zwanego Fibonaccim, odgrywa po dziś dzień doniosłą rolę w dociekaniach matematycznych, a rozważaniu jego własności poświęcono wiele dzieł. Odnosi się on, na przykład, do zjawisk przyrody. Liczba gałęzi drzewa w kolejnych latach tworzy ciąg Fibonacciego.

Pierwsze dwie liczby ciągu - to 1 i 1, każda następna jest sumą dwóch poprzednich. Oto słowo realizujące ten proces:

```
: FIBO 0 1 ROT 0 CR DO DUP 10 .R CR SWAP OVER + LOOP ;
```

Właściwością ciągu Fibonacciego jest to, że rośnie coraz szybciej. Dlatego już trzydziesta trzecia liczba, jaką wygeneruje nasz program będzie ostatnią w granicach liczb pojedynczej długości ze znakiem. Zastąpienie słowa . przez U. pozwoli uzyskać tylko jedną dalszą liczbę. Spróbujmy jednak posłużyć się liczbami podwójnymi:

```
:  DFIBO        ( n --- )
   0.           ( d=0 ) ( od początku używamy liczb podwójnych )
   ROT          ( 0. n ) ( przenosi górną granicę nad 0. )
   1. ROT       ( 0. 1. n ) ( taki sam manewr )
   0 CR         ( 0. 1. n 0 ) ( granice pętli ustalone )
DO              ( 0. 1. ) ( granice pętli przeniesione na stos )
                ( powrotów )
   2DUP         ( 0. 1. 1. )
   14 D.R       ( 0. 1. ) ( pierwsza liczba wydrukowana )
2SWAP           ( 1. 0. )
2OVER           ( 1. 0. 1.    )
      D+        ( 1. 1. ) ( pierwsze z kolejnych dodawań )
 LOOP ;         ( powrót do początku pętli )
```

Przy pierwszym powrocie na szczycie stosu znajdują się liczby podwójne 1 i 1, przy następnym będą to 1 i 2 itd. DFIBO pozwoli obliczyć 46 liczb Fibonacciego. Ostatnia, dziesięciocyfrowa 1 836 311 903 znajdzie się przy kolejnej granicy: liczb podwójnych ze znakiem. Pogoń za liczbami Fibonacciego nie jest, jak widać, łatwa.

Powróćmy do informacji o pętli liczonej. Istnieje słowo LEAVE, które pozwala ją opuścić.

Jeżeli, na przykład, chcemy, by kwadraty liczb były drukowane tylko do granicy liczb pojedynczej długości ze znakiem, możemy zastosować następującą konstrukcję:

```
: KWADRAT2 DUP * ;

: KWADRAT3 DO I KWADRAT2 DUP 32760
  IF DROP LEAVE THEN .  LOOP ;
```

Słowo to drukować będzie kwadraty liczb tylko wtedy, gdy nie będą większe od 32767. Zwróćmy uwagę, że przed sprawdzeniem warunku musimy skopiować badaną liczbę z pomocą DUP. Jak wcześniej była mowa, efektem porównania staje się znacznik logiczny, który z kolei zostanie "zużyty" przez słowo IF. Gdybyśmy nie skopiowali liczby, nie byłoby czego wydrukować.
Wielu programistów sceptycznie odnosi się do słowa LEAVE. Zwraca się uwagę, że po jego wystąpieniu pętla mimo wszystko musi być dokończona oraz że zdarzają się nieoczekiwane skutki działania LEAVE. Także w naszym przykładzie trzeba było obniżyć z tego powodu granicę dla porównań. Ponadto stwierdza się, że stosowanie LEAVE jest raczej następstwem nieumiejętnego użycia pętli liczonej zamiast innych struktur, które omówimy dalej, Mimo to w szeregu poważnych zastosowań LEAVE wykorzystywane jest do "awaryjnego" wyjścia z pętli, gdy kontynuowanie przewidzianych nią działań może wywołać ujemne skutki, na przykład, przy obsłudze stacji dyskietek.

FORTH rozporządza także drugą pętlą liczoną:

```
DO ... n  +LOOP
```

Liczba n poprzedzająca +LOOP oznacza krok wskaźnika czyli działa podobnie jak liczba przy STEP w pętli BASIC-u. Na rysunku 10 w miejscu oznaczonym "KROK" +LOOP zwiększa wskaźnik pętli o n, a LOOP - zawsze o 1.

Przykłady:

```
: TEST1 51 1 DO I . 2 +LOOP ;

: TEST2 0 20 DO I . -1 +LOOP ;
```

TEST1 wydrukuje tylko liczby nieparzyste, co może być czasem potrzebne. TEST2 pokazuje metodę zamiany granic pętli w porównaniu do DO...LOOP. Teraz "górną" granicą stało się zero i test wydrukuje kolejne liczby od 20 do 1, ale bez 0.

5.6 Przykład: tabliczka mnożenia

Jak stworzyć na ekranie tabliczkę mnożenia? Każdą z kolejnych liczb od 1 do 9 należy pomnożyć przez liczby od 1 do 9. Nasuwa się możliwość zagnieżdżenia drugiej pętli i mnożenia wskaźników obu pętli. Istnieją metody odczytywania tych wskaźników ze stosu powrotów, zastosujemy jednak najpierw prostszą: będziemy przechowywać kolejne wskaźniki pętli zewnętrznej w zmiennej.

Najpierw trzeba zdefiniować tę zmienną:

```
0 VARIABLE AA
```

Zacznijmy od oczyszczenia ekranu. Można to osiągnąć z pomocą odpowiedniego znaku sterującego, tzn. nie przeznaczonego do bezpośredniego drukowania. Oto kod znaku CLEAR w trzech komputerach:

ATARI 125

Commodore 147

Spectrum 251

W FORTH istnieje słowo EMIT drukujące znak, którego kod ASCII znajduje się na szczycie stosu. Na przykład

```
65 EMIT /Return/  A ok
```

Z pomocą tego słowa możemy wprowadzać również wszystkie znaki sterujące, w tym CLEAR.
Nasz przykładowy program powinien mieć tytuł, a w nagłówku i przy każdym wierszu odpowiednie liczby od 1 do 9. Pamiętać trzeba o odstępach między liczbami, co zapewnia .R, oraz o przechodzeniu do nowego wiersza z pomocą CR.

```
0 VARIABLE AA
: TABLICZKA 125 EMIT          patrz uwagi powyżej
CR CR 8 SPACES                nowy wiersz i przesunięcie kursora
." TABLICZKA MNOZENIA"        tytuł
CR 3 SPACES
```

```
10 1 DO I 3 .R LOOP CR          nagłówek

30 0 DO ." _" LOOP CR           podkreślenie

10 1 DO I DUP AA !              wskaźnik zewnętrznej pętli
                                wprowadzamy do zmiennej AA
. ." |"                         kopię drukujemy, pionową kreskę

10 1 DO I AA                    wskaźnik wewnętrznej pętli mnożymy
                                przez AA
3 .R                            drukujemy wynik mnożenia

LOOP                            koniec wewnętrznej pętli

CR                              nowy wiersz

LOOP                            koniec zewnętrznej pętli

;                               koniec definiowania słowa
```

W tym ćwiczebnym programie cztery razy wystąpiła pętla DO...LOOP, w tym trzy razy wykorzystany został wskaźnik I. Zwraca uwagę, że przy zagnieżdżeniu pętli interpretator nie pomylił dwóch wskaźników I. W niektórych systemach istnieją wskaźnik J i K pobierające wartości wprost ze stosu powrotów dla kolejnych zagnieżdżonych pętli. Można jednak, jak wskazuje przykład, wykorzystać pośrednią zmienną. Zapewne nie ucierpi na tym czas wykonania, ponieważ J i K wymagają skomplikowanych manewrów między stosami. Słowa te często pomija podstawowy podsłownik fig-FORTH. Natomiast FORTH-83, czyli standard ustanowiony przez FORTH Standard Team, traktuje J jako słowo obowiązkowe.

J można zdefiniować następująco:

```
: J  R> R> R> R R#  !  >R >R >R R# @  ;
```

Jak widać, J wymaga m. in. przeniesienia trzech wartości ze stosu powrotów na stos główny, odczytania czwartej wartości i przesłania z powrotem trzech liczb. Zmienna R# przechowuje pozycję kursora w buforze klawiatury. Pamiętajmy, że J jest wskaźnikiem pętli wewnętrznej, Z jego wykorzystaniem nasz przykładowy program przybierze postać następującą:

```
: TAB2 125 EMIT CR CR 8 SPACES ." TABLICZKA MNOŻENIA"
CR CR 3 SPACES 10 1 DO I 3 .R LOOP CR
10 1 DO I . ." |"
10 1 DO I J * 3 .R
LOOP CR LOOP ;
```

5.7 Pętle warunkowe: BEGIN...UNTIL i BEGIN. ..WHILE...REPEAT

Często zdarza się, że programując powtarzanie określonego ciągu działań nie wiemy, jaka będzie liczba powtórzeń, natomiast chcemy uzależnić ją od niespełnienia bądź spełnienia jakiegoś warunku.

Przypomina to dwa rodzaje sytuacji życiowych. Odkładamy pieniądze na nowy telewizor tak długo, jak długo nie jest spełniony warunek kupna: zebranie odpowiedniej kwoty. Kąpiemy się w rzece tak długo, jak długo spełniony jest warunek kąpieli, którym jest dostatecznie ciepła woda. Różnica polega na tym, że stan książeczki oszczędnościowej sprawdzamy po kolejnej wpłacie, natomiast temperaturę wody, jeżeli nie chcemy się przeziębić, przed kąpielą.

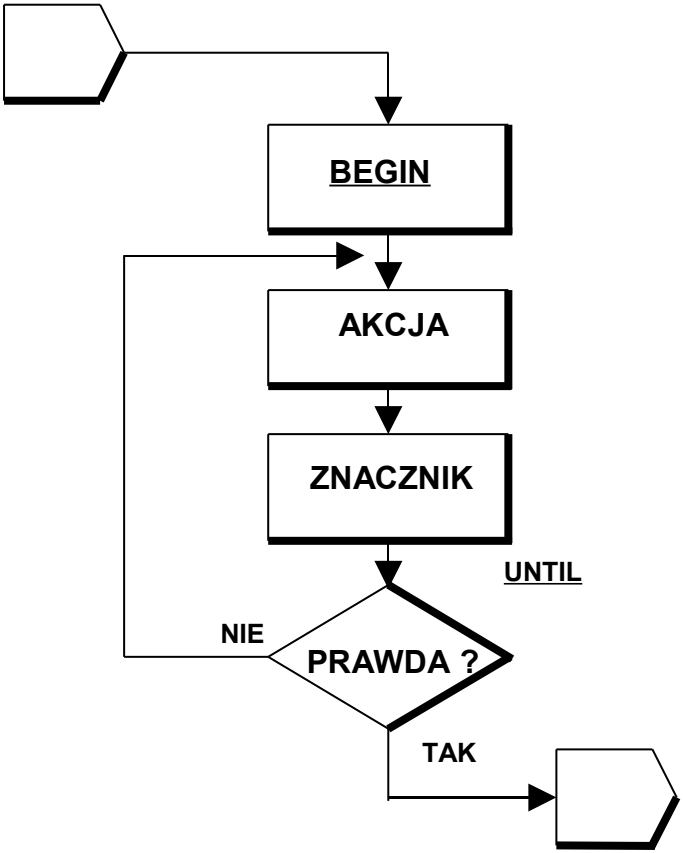
Na obie takie sytuacje FORTH znajduje odpowiedź w postaci dwóch niezmiernie efektywnych struktur: pętli sterowanych warunkiem, czyli mówiąc skrótowo warunkowych.
Pętla

```
BEGIN ...UNTIL
```

powoduje powtarzanie działań wtedy, gdy warunek nie jest spełniony i sprawdzenia dokonuje na końcu kolejnej sekwencji działań. Oznacza to, że w chwili, gdy ciąg działań dotrze do słowa UNTIL, powrót do BEGIN nastąpi tylko wtedy, gdy UNTIL na szczycie stosu napotka zero. Równoważnikiem UNTIL jest END, zwykle oba słowa są w słowniku.

Należy ponownie przypomnieć, że zgodnie z zasadami sprawdzania stosowanymi we wszystkich konstrukcjach warunkowych FORTH-a, owo zero zostanie przez UNTIL "zużyte" czyli usunięte ze stosu. Gdy natomiast UNTIL napotka i skasuje

wartość niezerową, nastąpi przejście za UNTIL i zakończenie pętli. Ukazuje to schemat blokowy na rysunku 11.



Rys. 11 Schemat blokowy BEGIN...UNTIL

Przykład:

```
: IKSY BEGIN ." X" 1 - DUP 0= UNTIL ;

4 IKSY XXXX ok
```

X było drukowane, dopóki (until) zmniejszanie liczby o 1 nie doprowadziło do przybrania przez nią wartości zerowej. Dopiero wtedy 0= umieściło na stosie wartość 1, co stało się dla

UNTIL sygnałem do zakończenia pętli. Widzimy tu takie przykład użytecznego zastosowania 0=.

A oto przykład przedstawienia klasycznego algorytmu Euklidesa dla znajdowania największego wspólnego dzielnika dwóch liczb. Metoda polega na tym, że ustalamy resztę z dzielenia większej liczby przez mniejszą, a następnie, obliczamy reszty z dzielenia mniejszej liczby przez poprzednią resztę i kontynuujemy takie dzielenia do chwili, gdy reszta wyniesie 0. Poprzednia reszta jest wówczas największym wspólnym dzielnikiem. Wykonuje to następujące słowo:

```
: NWP BEGIN SWAP OVER MOD -DUP 0= UNTIL ;
```

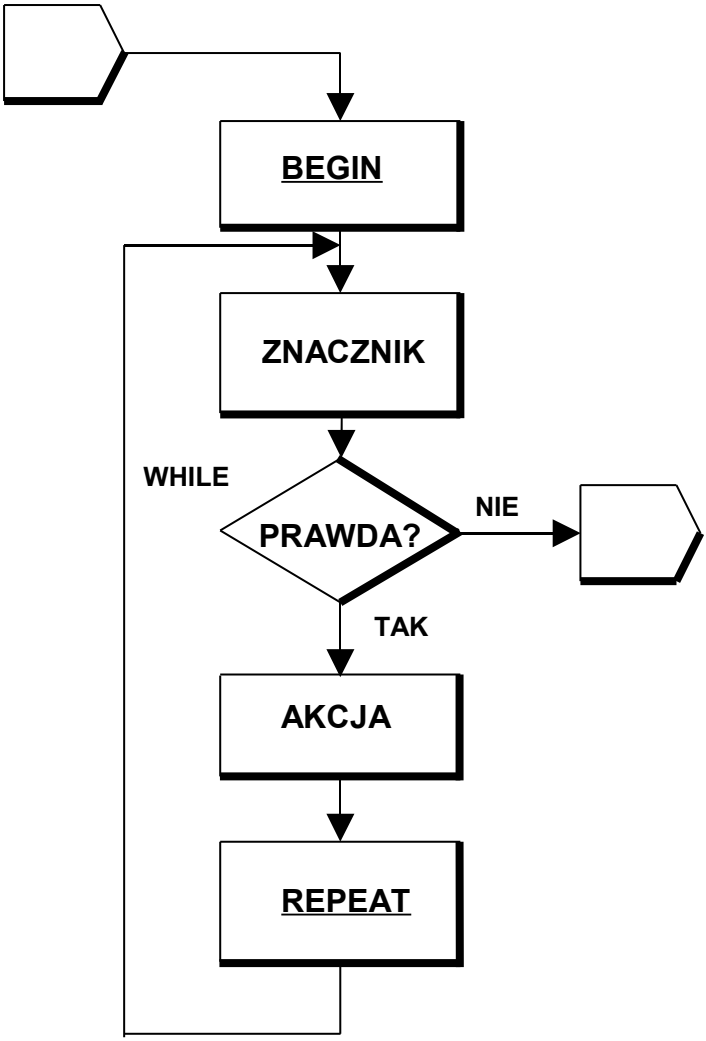
```
Np.   128 1024 NWP 126 ok

      324 556 NWP 4 ok

      1933 821 NWP 1 ok
```

BEGIN...WHILE...REPEAT - to drugi wariant pętli warunkowej. Przyjrzyjmy się jej działaniu posługując się schematem blokowym na rysunku 12, zamieszczonym na sąsiedniej stronie. słowem, które dokonuje kontroli spełnienia warunku, jest WHILE. Gdy WHILE napotka na stosie i skasuje zero, pętla kończy działanie i wykonanie przechodzi za REPEAT. Tak więc pętla ta może nie być wykonana ani razu, podczas gdy BEGIN... UNTIL zawsze wykonana jest co najmniej raz.

Gdy WHILE napotka na stosie i skasuje wartość niezerową, wówczas wykonana zostanie sekwencja między WHILE a REPEAT, po czym - zwróćmy na to uwagę - nastąpi powrót do początku struktury, czyli tuż za BEGIN. Tak więc omawiana struktura jest w porównaniu z poznanymi wcześniej znacznie bardziej "pojemna", bowiem w jej obrębie można wykonać zarówno ciąg działań prowadzących do obliczenia warunku, jak i działań następujących wtedy, gdy warunek jest spełniony. Umożliwia to w praktyce umieszczenie kontroli warunku kontynuowania pętli w dowolnie wybranym jej miejscu, w tym przed wejściem w nią. Obrazuje to rysunek zamieszczony na sąsiedniej stronie



Rys. 12 Schemat blokowy BEGIN...WHILE...REPEAT

BEGIN...WHILE...REPEAT jest, jak można sądzić, najbardziej przydatną konstrukcją programowania strukturalnego w FORTH, Umiejętność posługiwania się nią jest jednym z istotnych sprawdzianów opanowania programowania w tym języku. BASIC nie zawiera odpowiednika tej struktury i podobne zadania muszą być w nim realizowane z pomocą czasochłonnych skoków miedzy liniami.

Warto zapamiętać, że wszystkie omówione w tym rozdziale pętle mogą być zagnieżdżane na znaczną głębokość, ograniczoną jedynie pojemnością stosu powrotów oraz wielkością pamięci komputera.

5.8 Przykład: liczby pierwsze

Już w starożytności liczby pierwsze zaprzątały uwagę uczonych. Co decyduje o tym, że gdy większość z nich można podzielić przez niektóre inne bez reszty, są również takie, które można w ten sposób podzielić tylko przez nie same oraz przez 1? Te ostatnie nazwano liczbami pierwszymi. Badania nad nimi trwały w ciągu stuleci i rozwinęły się w ważny dział teorii liczb. Współcześnie wytwarzanie dużych ilości liczb pierwszych staje się niezbędne przy stosowaniu niektórych najnowocześniejszych technik obliczeniowych.

Postawmy sobie znacznie skromniejsze zadanie: zbudujmy program, który będzie wytwarzał liczby pierwsze z przedziału od 2 do 255 metodą wynalezioną 22 wieki temu przez Eratostenesa z Cyreny - znakomitego greckiego matematyka i astronoma. Jego innym osiągnięciem było wysoce dokładne obliczenie promienia Ziemi.

Na czym polega metoda wyszukiwania liczb pierwszych nosząca nazwę sita Eratostenesa? Liczby z określonego przedziału dzielimy kolejno przez 2,3 i liczby następne, przy czym za każdym razem usuwamy ("odsiewamy") liczby, które dzielą się bez reszty. Wystarczy to czynić do chwili, gdy kwadrat dzielnika stanie się większy od dzielonej liczby.

Sito Eratostenesa, wprawdzie w ulepszonych wersjach, stanowi również dziś podstawę algorytmów wyszukujących liczby pierwsze. Zastosujemy jego najprostszą wersję.

Tworzymy najpierw zmienną w postaci tablicy mającej 254 bajty. Każda z liczb w wybranym przez nas zakresie mieści się w jednym bajcie.

```
        0 VARIABLE ERATOS 254 ALLOT
```

Dwa następne słowa wprowadzą do tej tablicy liczby od 1 do 255, a potem zastąpią zerami wielokrotności kolejnych liczb n.

```
: WSTAW  256 1  DO I OVER C!  1+  LOOP  DROP  ;

: USUN                    ( n --- )
   256  OVER              ( n  256  n )
  DO                      ( n )
    I  1  ERATOS          ( n  I-1  adres )
    +                     ( n adres  ( I )   )
    0     SWAP     C!     ( n  )( wstawia 0 pod adres (I)
    DUP                   ( n n )
  +LOOP                   ( n )( I = I+n, powrót do początku )

DROP  ;
```

Ciekawym zastosowanym tu pomysłem, wziętym z pracy [5] , jest wyeliminowanie mnożeń i dzieleń. Zostały one zastąpione odpowiednim ustawieniem kroku +LOOP. Np. gdy n=5 pętla zaczynając od liczby 5 w tablicy usuwa dalej co piątą liczbę, pozostawiając jednak 5. Po tych czynnościach przygotowawczych budujemy słowa końcowe, które wykorzystuje stworzone mechanizmy do odsiania wszystkich liczb mających dzielniki inne niż 1 i one same.

```
: PIERWSZE  ( --- )
    ERATOS WSTAW CR
    ERATOS           ( adr )
    256  2           ( adr 256 2 )
  DO                 ( adr )
    I  1  -          ( adr I-1 )
    OVER             ( adr I-1 adr )
    +                ( adr adr+I-1 )
    C@               ( adr n )
    -DUP  IF
    DUP . USUN       ( adr )  ( wykonywane, gdy n<>0 )
          THEN
  LOOP
DROP  ;
```

Celowe jest uważne prześledzenie tego, co dzieje się na stosie w obrębie pętli DO...LOOP. Słowo -DUP kopiuje n tylko w przypadku, gdy nie zostało wcześniej wyzerowane i gdy n będzie potrzebne w obrębie konstrukcji warunkowej IF...THEN.
W przeciwnym wypadku, gdy n=0, zniknie ono i nastąpi skok wprost do LOOP, a stamtąd tuż za DO. Przy tym przeskoku licznik pętli wykorzystywany przez I zostanie zwiększony o 1. Zwróćmy uwagę, że powrót na początek pętli następuje przy zachowaniu na stosie niezmienionego adresu początkowego zmiennej ERATOS. Łatwo zauważyć, że będzie on ciągle przechowywany na stosie jakby "pod spodem" aż do wykonania ostatniego cyklu, po czym końcowe DROP usunie go i stos będzie pusty.

ROZDZIAŁ 6
PAMIĘĆ I EKRAN

Jednym ze źródeł dużej sprawności FORTH-a jest zastosowanie w nim skutecznych narzędzi operowania na blokach pamięci, wprowadzania, wyprowadzania i przemieszczania danych. Wiele słów FORTH-a służy do utrzymywania sprawnej dwukierunkowej łączności między słownikiem a stosami oraz między tymi dwu podstawowymi strukturami a pamięcią, ekranem i innymi urządzeniami zewnętrznymi.

6.1 Operacje na blokach pamięci

Podstawowym operatorem służącym do przemieszczania bloków pamięci jest CMOVE. Należy poprzedzić go wprowadzeniem trzech liczb, które kolejno oznaczają: adres początku przemieszczanego bloku, adres, od którego ma on być skopiowany oraz liczbę przesuwanych bajtów. CMOVE umożliwia łatwe wpisywanie do RAM dużych fragmentów ROM, na przykład tablicy znaki w celu redefiniowania części z nich. Słowo to jest z reguły zdefiniowane w języku maszynowym. Kopiuje bloki bardzo szybko.

Ważne jest następujące zastrzeżenie. CMOVE zaczyna, przemieszczanie od bajtu znajdującego się na początku odcinka i stopniowo przesuwa działanie ku coraz wyższym adresom. Jeżeli zatem odcinek docelowy znajduje się wyżej, a obszary częściowo się pokrywają, dochodzi do utraty części informacji. Nie ma takiej obawy, gdy przemieszczamy blok w dół pamięci lub gdy obszary źródłowy i docelowy nie nakładają się na siebie. Trudnościom można zapobiec przerzucając najpierw blok wysoko w górę, a potem wracając z nim na wyznaczone miejsce.

W niektórych realizacjach występują dodatkowe operatory powodujące, że przesuwanie zaczynu się od najwyższego adresu odcinka źródłowego.

Opisana własność CMOVE pozwala z jego pomocą zdefiniować słowo FILL powodujące wypełnienie określonego fragmentu pamięci jedną wartością, na przykład, zerem lub liczbowym odpowiednikiem spacji. FILL oczekuje na stosie adresu początkowego, liczby bajtów oraz wartości służącej do wypełnienia obszaru.

Pisząc

```
15000 1024 0 FILL
```

spowodujemy, że odcinek o długości kilobajta poczynając od adresu 15000 wypełni wartość 0 . Czynność tę wykonuje zresztą jedno z dwóch słów zdefiniowanych z pomocą FILL i z reguły dostępnych w podstawowym słowniku: ERASE i BLANKS.

```
: ERASE 0 FILL ;

: BLANKS 32 FILL
```

Oba słowa oczekują adresu i liczby bajtów, wprowadzanych na stos w takiej właśnie kolejności.

6.2 Wprowadzanie i wyprowadzanie ciągów znaków

Komputer zapisuje w pamięci tekst, który mu przekażemy, w postaci liczb kodu ASCII odpowiadających kolejnym znakom. Do wyprowadzania pojedynczego znaku służy słowo EMIT. Drukuje ono znak, którego kod znajduje się w mniej znaczącym bajcie liczby na szczycie stosu. Liczby na stosie są, jak wiemy, dwubajtowe, toteż np. 65 ma bardziej znaczący bajt równy zeru.

```
HEX 5B41 EMIT
```

wydrukuje A, ponieważ w dolnym bajcie jest 41 hex czyli 65 dec. Cała liczba będzie usunięta ze stosu.

Odwrotnie niż EMIT działa KEY z dodatkowym jeszcze efektem. Key czeka mianowicie na znak, po czym przekazuje na stos jego kod ASCII. Jeżeli napiszemy

```
KEY . (Return)
```

kursor zatrzyma się nieruchomo. Gdy teraz naciśniemy klawisz "C", w tym samym wierszu zostanie wpisane 67. W ten sposób z pomocą KEY można łatwo ustalić kod ASCII każdego znaku.

Podobnie działa dostępne w niektórych implementacjach słowo ASCII, tyle że nie daje efektu oczekiwania na znak, natomiast ten, który wpiszemy, umieszcza na szczycie słownika. Znaczenia ASCII mogą być zresztą rozmaite.
Właściwości KEY pozwalają wykorzystać to słowo tak, aby po naciśnięciu dowolnego klawisza, komputer drukował jakiś napis. Na przykład:

```
CR KEY DROP ." WITAJ"
```

po naciśnięciu jakiegokolwiek klawisza wydrukuje w następnej linii "WITAJ".

Do wyprowadzania na ekran lub inne urządzenie ciągu znaków służy słowo TYPE. Na stos należy wprowadzić adres początkowy, a potem liczbę bajtów, czyli takie same dane, jak przy DUMP. Ciągi znaków przechowywane są często w pamięci z bajtem długości na początku. Istnieje słowo COUNT, które w przypadku tak przygotowanych ciągów wprowadza na stos po podaniu adresu ciągu adres o 1 większy oraz zawartość bajtu długości przygotowując tym samym dane dla TYPE. Pełna definicja TYPE ilustruje stosowanie wielu poznanych już słów:

```
: TYPE -DUP IF OVER + SWAP DO I C@  EMIT LOOP
ELSE DROP THEH ;
```

Trzonem definicji jest pętla wyprowadzająca z pomocą EMIT kolejne znaki. TYPE drukuje wszystko, łącznie ze spacjami. Istnieje słowo -TRAILING umożliwiające pominięcie w druku spacji znajdujących się na końcu wyprowadzanego tekstu.
FORTH rozporządza również wygodnym mechanizmem wprowadzania ciągów znaków z klawiatury, do czego służą słowa EXPECT, QUERY i WORD, przy czym każde z nich pełni nieco odmienne funkcje.

EXPECT po otrzymaniu adresu i liczby znaków czyta je z terminalu i umieszcza pod wskazanym adresem w pamięci. Jeżeli nie wiemy, ile znaków wprowadzimy, można ustalić ich liczbę z nadmiarem, a wprowadzenie zakończyć klawiszem Return, który kończy działanie EXPECT.
EXPECT wykorzystywane jest przez system operacyjny FORTH-a do czytania słów komend z ważnego ogniwa systemu, jakim

jest bufor klawiatury. Adres początkowy tego buforu przechowywany jest w zmiennej użytkownika TIB, co jest skrótem angielskiej nazwy Terminal Input Buffer.

Pokrewne w stosunku do EXPECT jest działanie słowa QUERY. Zasadnicza różnica polega na tym, że QUERY nie umieszcza wprowadzanego ciągu znaków w pamięci, lecz przechowuje go w buforze terminalu, skąd znaki mogą być pobrane do różnych celów. QUERY nie wymaga żadnych danych wstępnych. Czeka ono, aż wprowadzimy linię złożoną z maksymalnie 80 znaków, umieszcza te znaki a ściślej ich kody ASCII poczynając od adresu zawartego w TIB i ustawia na zero wewnętrzny licznik buforu terminalu, jakim jest zmienna IN. Widoczne to jest z definicji QUERY wykorzystującej omówione przed chwilą słowa :

```
: QUERY TIB @ 80 EXPECT 0 IN ! ;
```

Wydawałoby się, że QUERY (po angielsku oznacza to po prostu: znak zapytania) może być użyte w programach, które wymagają wprowadzenia napisów z klawiatury, np. dość częstego w grach podania imienia i nazwiska. Spróbujmy:

```
: NAZWISKO? CR ." PODAJ SWOJE NAZWISKO: " QUERY ;

NAZWISKO?

PODAJ SWOJE NAZWISKO: PIOTR KLEKS PIOTR?
```

Sygnał błędu wynikł z tego, że wszystko, co wprowadzamy do buforu klawiatury, interpretator poddaje weryfikacji. Ponieważ słowa PIOTR nie ma w. słowniku, sygnalizowany jest błąd.

Możemy jednak rozwiązać nasz problem z pomocą innego słowa: WORD. Może być ono użyte do wielu celów. Najważniejszym jest czytanie nazw nowych wejść słownikowych. Jest to możliwe dlatego, że w większości systemów WORD pobiera ciąg znaków z buforu klawiatury i umieszcza na szczycie słownika, czyli pod adresem podawanym przez HERE, przy czym poprzedza znaki bajtem długości, tak jak jest to potrzebne przy definiowaniu słowa. Ważną cechą WORD jest to, że może czytać znaki również z buforów urządzeń zewnętrznych.

WORD czyta znaki aż do napotkania ogranicznika, którym może być dowolny znak. Jego kod musimy podać przed WORD. Jak odróżnia WORD, czy trzeba czytać znaki z klawiatury czy np. z buforu stacji dyskietek? Służy do tego zmienna BLK. Gdy jest ona równa zeru, tekst czytany jest z buforu terminalu, w przeciwnym wypadku - z bloku, którego adres zawiera BLK.

Dotknęliśmy tu bardziej skomplikowanych cech funkcjonowania FORTH-a, które będą omówione w drugiej części książki. Wróćmy do naszej nieudanej dotychczas próby podania nazwiska z klawiatury. Zbudujmy nową definicję:

```
: NAZWISKO? CR ." PODAJ SWOJE NAZWISKO: " QUERY 155 WORD HERE
```

Kod 155 odpowiada w ATARI naciśnięciu klawisza Return. Liczba ta jest rozmaita w różnych komputerach i w trzech najpopularniejszych wynosi:

Atari 155

Commodore 13

Spectrum 254

Gdy wypróbujemy nowo zdefiniowane słowo NAZWISKO? uzupełnione o WORD, przekonamy się, że działa poprawnie. W niektórych systemach WORD daje na stos adres, pod którym w danej chwili znajduje się wpisane nazwisko. Teraz wydrukuje nasze nazwisko

```
: DRUKUJ NAZWISKO? CR COUNT TYPE ;
```

6.3 Wprowadzanie liczb

FORTH nie ma specjalnego słowa służącego do wprowadzania liczb z klawiatury. Wiemy, że wystarczy po prostu liczbę napisać. Można jednak w oparciu o poznane dotychczas słowa zdefiniować operator pełniący funkcję analogiczną do tej, jaką odgrywa w BASIC-u komenda INPUT. Słowo INPUT będzie, czekać na liczbę, którą zechcemy wprowadzić, po czym umieści ją na stosie.

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

Obok QUERY i WORD mamy tu jeszcze nie poznane słowo NUMBER. Przekształca ono ciąg znaków poprzedzony bajtem długości w liczbę podwójną. Jeżeli znaki nie są cyframi, powstaje błąd. Widzimy tu, że WORD przygotowuje ciąg cyfr w takiej postaci, jakiej wymaga NUMBER. Końcowe DROP usuwa górną część liczby podwójnej. Oczywiście, możliwe jest skonstruowanie słowa działającego analogicznie w przypadku liczb podwójnych.

FORTH rozporządza również dwu słowami pozwalającymi umieszczać wprowadzane liczby wprost na szczycie słownika:

, n --- umieszcza tam liczbę pojedynczej długości

C, c --- umieszcza na szczycie stosu liczbę jednobajtową.

Działanie tych słów omówimy szerzej w rozdziale 9 poświęconym kompilacji. Oba zwiększają odpowiednio wskaźnik słownika DP, a zatem również HERE.

Ćwiczenie 10

Zbudujmy słowo DINPUT do wprowadzania liczb podwójnej długości.

ROZDZIAŁ 7
JAK ZBUDOWANY JEST FORTH?

7.1. Kilka uwag wstępnych

Języki programowania wysokiego poziomu powstały i rozwijają się przede wszystkim w tym celu, by odciążyć programującego od wysiłku, jakiego wymaga programowanie w języku maszynowym. Pozwalają wyrazić skomplikowane procesy dziejące się we wnętrzu maszyny w terminologii zbliżonej do mowy potocznej. Znajduje to wyraz chociażby w fakcie, że komendy języków wysokiego poziomu mają nazwy opisujące, zazwyczaj po angielsku, czynności podobne do tych, z jakimi stykamy się w życiu codziennym. Komendy w BASIC są tego dobitnym przykładem. PRINT - drukuj, INPUT - wprowadź, SOUND - dźwięk, POSITION - pozycja, LOAD - ładuj. Również większość słów FORTH-a dla osób znających język angielski brzmi całkiem swojsko. Wydawało by się zatem, że ktoś, kto poznał język programowania wysokiego poziomu, zostaje całkowicie odciążony od konieczności poznania języka samej maszyny, że nie musi Interesować się, co dzieje się w jej wnętrzu. Postawa taka pojawiła się w toku rozwoju języków programowania i znalazła, np. wyraz w dość barwnym stwierdzeniu Glenforda J. Myersa: "Gdybym był kierownikiem działu przetwarzania danych, wszedłbym do mojego ośrodka obliczeniowego i fizycznie wyłączył wszystkie asemblery" [12, s. 137].
Rozwój informatyki i coraz bardziej masowe stosowanie mikrokomputerów doprowadziły do skorygowania tego poglądu. Okazywało się nieraz, że dla pewnych zadań brakuje po prostu dostatecznie efektywnych narzędzi w językach programowania wysokiego poziomu. Z krytyki tego stanu zrodził się zresztą FORTH. Jego twórca pisał [1]: "Małe komputery nie mogą być skutecznie zaprogramowane z pomocą języka wysokiego poziomu". Myślał tu, oczywiście, o językach wtedy istniejących. Można i jego pogląd uznać w jakiejś mierze za skrajny. Faktem jest jednak, że w programach pisanych w wielu językach z reguły stykamy się z wstawkami w języku maszynowym oraz że także mało zaawansowanemu programiście przydaje się nieraz znajomość mapy pamięci komputera i niektórych ważnych w niej

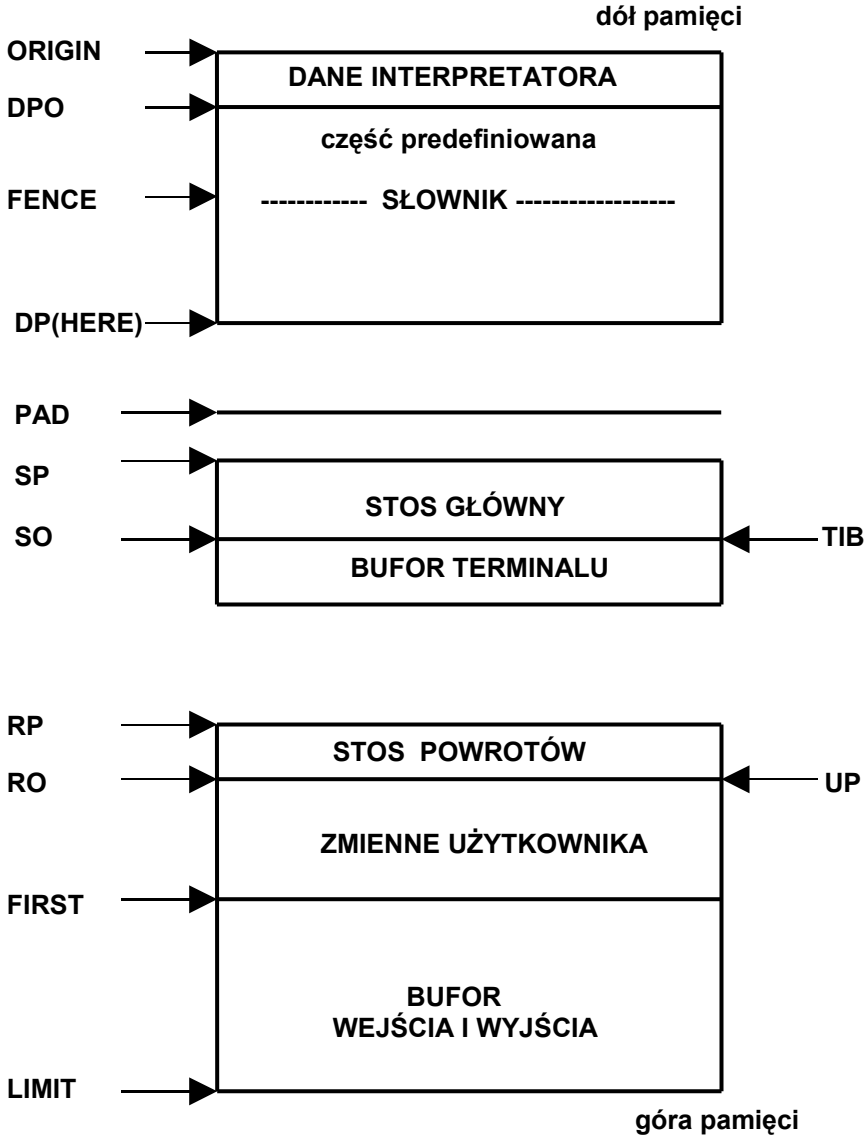
adresów. PEEK i POKE, choć brak ich w pierwotnej wersji BASIC, zdobyły sobie prawo obywatelstwa, a ukrywanie wstawek w Języku maszynowym w liniach DATA jest w tym języku nagminne. FORTH pozwala uniknąć takich wstawek, bowiem asembler nie jest w nim "obcym ciałem", lecz integralną częścią systemu. Z drugiej jednak strony będąc językiem wysokiego poziomu zachęca nas wprost, byśmy lepiej poznawali maszynę i jej język oraz to, co dzieje się w jej wnętrzu, gdy tworzymy i wykonujemy program.

Przemyślana i logiczna struktura wewnętrzna FORTH, o której mieliśmy możność nieraz już wspomnieć, jest jego mocnym atutem i w poważnej mierze źródłem jego efektywności. Poznanie i zrozumienie jej - to istotny krok naprzód w rozwijaniu umiejętności programowania. Nie jest zatem przypadkiem, że wszyscy, którzy starają się w systematyczny sposób przedstawić FORTH, opisują jego mapę pamięci. Uczynimy to również.

7.2 Mapa pamięci

Pragnąc stworzyć język jak najbardziej przydatny do wykonywania dużych zadań na małym komputerze Charles H. Moore podporządkował temu celowi również zasady rozmieszczania FORTH w pamięci. zajmuje on mianowicie dość zwarty blok w dolnej części pamięci operacyjnej pozostawiając całą resztę dla rozszerzającego się ku górze, czyli ku wyższym adresom pamięci, programu użytkownika. Przedstawiona na rys. 13, mapa pamięci FORTH dobitnie to ilustruje. Jest to rozwiązanie ramowe, ulegające w niektórych komputerach pewnym zmianom dyktowanym przede wszystkim specyficznymi cechami ich architektury. Np. w Atari i innych komputerach opartych na mikroprocesorze 6502 stos główny umieszczony jest, na części strony zerowej. Pozwoliło to na posłużenie się w zdefiniowanych w języku maszynowym pierwotnych słowach FORTH specjalnym, dostępnym w 6502 trybem adresowania na stronę zero, znacznie zwiększającym szybkość przetwarzania danych. Możliwości takich nie ma, np. w Spectrum, w którym strona zerowa zajęta jest przez ROM.

Organizacja pamięci Atari i innych komputerów o pokrewnej architekturze umożliwia ponadto umieszczenie również innych składników struktury FORTH, Jak bufory wejścia/wyjścia, bufor terminalu i stos powrotów poniżej słownika, dzięki czemu prościej przebiega jego rozszerzanie się ku górze. Nie jest też



Rys. 13 Mapa pamięci FORTH-a

zapewne przypadkiem, że FORTH Interest Group posłużyła się kodem 6502 dla zdefiniowania maszynowej części słów pierwotnych języka w swym podręczniku jego implementacji.

Należy zarazem podkreślić, że różnice rozwiązań szczegółowych w niczym nie naruszają cennej właściwości FORTH, jaką jest to, że jego podstawowe części składowe we wszystkich implementacjach są takie same, że, co więcej, są takie, jakimi przy narodzinach języka zaprojektował je Moore.

7.3 Główne części składowe

Przyjrzyjmy się mapie pamięci na rys.13. Przedstawia ona wszystkie składniki FORTH i ich rozmieszczenie w pamięci. Ponadto na krawędziach poszczególnych części wypisane są słowa, w większości zmienne użytkownika, choć nie tylko one, zawierające adresy miejsc, przy których znajdujemy je na rysunku. W poszczególnych wersjach FORTH może brakować niektórych słów. Niejednokrotnie, jak w przypadku SO, łatwo jest je zdefiniować, kiedy indziej brak zmiennej wynika z założenia, że nie powinna być ona dostępna dla użytkownika. Dotyczy to zwłaszcza stosu powrotów - struktury wyjątkowo wrażliwej na błędne ingerencje, które, z reguły powodują zawieszenie się, czyli unieruchomienie programu i konieczność posłużenia się klawiszem Reset lub nawet ponownego wgrywania interpretatora.

Brakuje nieraz także wskaźnika początku słownika DPO, łatwo go jednak ustalić, ponieważ jest to adres początku pierwszego słowa, czyli tego, które VLIST wyświetla jako ostatnia. W fig-FORTH jest to słowo LIT odgrywające, podobnie jak CLIT, podstawową rolę w kompilowaniu liczb podczas definiowania słów. Przejrzyjmy teraz kolejno składniki mapy pamięci od góry rysunku, tzn. w rzeczywistości od najniższych adresów w pamięci.

Pierwsze pole, "dane interpretatora", zawiera zestaw podstawowych wielkości, które bezpośrednio po wgraniu i inicjalizacji intetpretatora zostają umieszczone jako wartości wyjściowe w głównych zmiennych systemu. Obszar ten zaczyna się od maszynowego rozkazu skoku do słowa-procedury COLD wykonującego tzw. zimny start. Przechowuje się tu również szereg ważnych dla systemu adresów, np. adres ostatniego słowa w słowniku i jego szczytu, czyli wartości, które w chwili rozpoczęcia pracy powinny być nadane zmiennym LATEST i DP.

Początek obszaru danych ustalamy z pomocą słowa +ORIGIN.

Pisząc 0 +ORIGIN .

odczytujemy adres, pod którym zaczyna się FORTH. Obszar danych interpretatora jest z reguły bardzo mały, np. w Extended fig-FORTH zajmuje 38 bajtów.

Kolejny obszar - to podstawowa struktura FORTH: jego słownik. Rośnie on w górę pamięci, a jego rozmiary ogranicza jedynie pamięć komputera. W słowniku wyodrębnia się jego dolna część, na którą składają się słowa predefiniowane czyli skonstruowane przez twórców interpretatora. Część z nich, zwłaszcza znajdujących się na początku słownika, zostało zdefiniowanych w języku maszyn i nosi nazwę słów pierwotnych [Primitives]. Budowy tych słów nie możemy odczytać z pomocą instrumentów FORTH, a jedynie w asemblerze. Natomiast zdecydowana większość słów predefiniowanych opracowana została już w FORTH i ich budowa ściśle odpowiada tej, jaką opisaliśmy w rozdziale 4. Część słów posługuje się wstawkami w asemblerze FORTH, o którym opowiemy w rozdziale 11.

Granicę między dwu obszarami słownika wyznacza zmienna FENCE, Zawiera ona adres najwyższego słowa obronionego przed skasowaniem z pomocą FORGET. Jeżeli spróbujemy skasować np. słowo HERE odpowiedzią będzie komunikat o błędzie: "In protected Dictionary" Elastyczność FORTH wyraża się jednak również w tym, że użytkownik może sam ustalić położenie FENCE i tym samym albo zwiększyć obszar słów obronionych, albo go zmniejszyć i skasować dowolną część FORTH, na własne ryzyko. Aby skasować definicje chronione, można postąpić następująco. Wybieramy słowo, które jako ostatnie chcemy zachować w słowniku, powiedzmy XXX i piszemy

XXX FENCE ! /Return/

Od tej chwili możemy z pomocą FORGET skasować każdy fragment słownika powyżej nowo ustalonej granicy FENCE. Metoda ta jest użyteczna w systemach, które, jak Extended fig-FORTH, umożliwiają ponowne zapisanie na dyskietkę całego znajdującego się w pamięci słownika łącznie ze słowami przez nas stworzonymi i automatycznie nakładają ochronę FENCE na całość. W innym przypadku stosownie tej metody wymaga umiejętności przebudowania słownika, zastępowania części predefiniowanych słów nowymi. Może to być wdzięcznym zajęciem dla miłośników

eksperymentowania. Nic przecież nie stoi na przeszkodzie temu, byśmy wynaleźli coś, czego nikt dotychczas nie wymyślił.

Pierwszy adres nad szczytem słownika zawiera zmienna DP, którą już poznaliśmy. Miejsce to jest, jak wiemy, ruchome. Obniża się przy kasowaniu słów, a podwyższa, gdy rozbudowujemy słownik. Dlatego nad słownikiem znajduje się obszar wolnego miejsca. Gdy słownik rośnie, wszystkie elementy znajdujące się nad nim przesuwają się w górę pamięci. System wykonuje to automatycznie.
Kreską zaznaczyliśmy umownie miejsce, które podaje zmienna PAD. Znajduje się ono zazwyczaj w odległości 128-132 bajtów od szczytu słownika, toteż PAD można zdefiniować następująco:

```
: PAD HERE 127 + ;
```

Znajduje się tu rodzaj buforu po angielsku zw. scratchpad służącego do czasowego przechowywania tekstów, a także do formatowanego wyprowadzania liczb, o którym powiemy w rozdz. 10. Scratchpad można by przetłumaczyć jako "ochraniacz", "podkładka startowa". Nazwa ta powstała ze względu na to, że w podstawowej konfiguracji bufor ten rozdziela słownik i stos główny.

Następną pozycję na rysunku zajmuje stos główny. Rośnie on w dół czyli w przeciwnym kierunku niż słownik, a podstawę ma u góry. Tak więc rzeczywisty kierunek wzrostu stosu jest odwrotny w stosunku do terminologii, jaką się posługujemy. Nie stanowi to, rzecz prosta, żadnej przeszkody dla posługiwania się pojęciami: szczyt stosu, góra stosu itd. Natomiast odwrotny kierunek rośnięcia stosu ma ważną zaletę: liczby na nim mają dokładnie taką samą kolejność, jak w pamięci komputera.

Pierwszą nie zajętą lokację na stosie wskazuje zmienna SP, a odczytać ją można z pomocą słowa SP@ . Stos główny jest strukturą ważną, lecz niewielką. 64 bajty traktuje się jako wielkość wystarczająca. Może więc on pomieścić do 32 liczb pojedynczej i do 16 podwójnej długości. Zdarzają się jednak stosy mniejsze. Przepełnienie stosu, podobnie jak próba pobierania liczby z pustego stosu, powoduje błąd. Stos nie służy do długotrwałego przechowywania liczb. Na to mamy stałe i zmienne, ich tablice i wiele innych form. Zasada, że stos rośnie w dół, przestrzegana jest również przy innym jego umieszczeniu w pamięci. Obowiązuje zresztą z reguły również we wbudowanych stosach komputera.

W bezpośrednim sąsiedztwie ze stosem głównym

umieszczony jest w sposób bezkolizyjny bufor terminalu klawiatury wypełniany od początkowej komórki, której adres zawiera zmienna TIB. Działanie tego buforu omówiliśmy. Ma on zwykle pojemność 128 bajtów.

Kolejne ważne ogniwo systemu to stos powrotów. Jego granice zawarte są w zmiennych RP i RO, nie zawsze dostępnych dla użytkownika. Zgodnie z ogólną zasadą stos powrotów rośnie w dół. Za dostateczne uważa się, gdy ma 48 bajtów.

Następny odcinek to obszar zmiennych użytkownika, o których znaczeniu i zastosowaniach mówiliśmy. Początek obszaru zawiera wskaźnik UP (user pointer), a rozmiary określone są przez system i użytkownik ma na nie nikły wpływ.

Ostatnie wreszcie pole mapy pamięci FORTH obejmuje bufory wejścia/wyjścia. Dolna i górna granica ich obszaru podawana jest odpowiednio w zmiennych FIRST i LIMIT. W systemach, w których, jak w Atari, bufory znajdują się poniżej słownika, FIRST i LIMIT są stałymi, czyli podają wprost wartość.

Bufory wejścia/wyjścia mają taką samą wielkość jak ekran na dyskietce lub taśmie, czyli każdy z nich mieści na ogół 1024 bajty, chociaż są również wersje, w których zachowano stosowane dawniej mniejsze rozmiary buforów. W typowym układzie są trzy bufory. Odpowiada temu słowo TRIAD, umożliwiające wyprowadzenie trzech kolejnych ekranów, co jest wygodne przy ich drukowaniu. Często jednak są tylko dwa bufory wykorzystywane na zmianę.

C Z Ę Ś Ć II
ZASTOSOWANIA

ROZDZIAŁ 8
KOMPILACJA

8.1 Posłowniki

W obrębie słownika FORTH-a możliwe jest definiowanie dowolnej liczby podsłowników. W języku angielskim rozróżnienie to podkreślana jest użyciem dwóch odmiennych słów:

dictionary - słownik i vocabulary - podsłownik.

Możliwość, tworzenia podsłowników z wielu powodów jest istotna. Pozwala wyodrębnić pewne fragmenty słownika i nadać mu tym samym strukturę blokową. Istotniejsze są względy praktyczne. Wywołując podsłownik z pomocą jego nazwy powodujemy, że zarówno w fazie definiowania nowych słów, jak i podczas natychmiastowego wykonywania słów, interpretator zaczyna poszukiwania najpierw we wskazanym przez nas podsłowniku, co przyspiesza tę czynność. Nie dotyczy to słów skompilowanych, które, jak mówiliśmy wcześniej, mają kod gotowy do natychmiastowego wykonania.

Udogodnieniem jest również to, że podział na podsłowniki pozwala używać w nich słów o takich samych nazwach, lecz odmiennym znaczeniu. Wprawdzie interpretator FORTH-a w czasie wprowadzania takiego słowa będzie sygnalizować błąd: Isn't unique - nie jest jedyne, ale wykonanie nie będzie zakłócone, jeżeli wyraźnie określimy, z którego podsłownika wywołujemy słowo. Na przykład, w podstawowym podsłowniku jest słowo I, ale również podsłownik edytora z reguły zawiera I, tyle że w tamtym przypadku powoduje ono wprowadzenie nowej linii między już zapisane.

Podstawowy podsłownik FORTH-a zawierający słowa predefiniowane oraz te, które będziemy definiować bez wskazania odrębnego podsłownika, nosi nazwę taką samą, jak język: FORTH.

Dwa dalsze ważne podsłowniki - to ASSEMBLER i EDITOR. Wywołując podsłownik przez napisanie jego nazwy spowodujemy, że zawarte w nim słowa po VLIST wypisane zostaną na samym początku listy nazw słów. Natomiast w słowniku podstawowym znajdują się zawsze nazwy podsłowników. Jeżeli na napisanie ASSEMBLER otrzymamy odpowiedź w postaci sygnału o błędzie, oznacza to, że tego podsłownika nie ma w pamięci. Może się on natomiast znajdować na dyskietce lub taśmie i powinien być stamtąd wgrany. Jeżeli i tam go nie ma, pozostaje jeszcze jedna droga: zbudować go samemu. W odniesieniu do asemblera i edytora mowa będzie o tym w rozdziale 11.

Dwie zmienne użytkownika CONTEXT i CURRENT pomocne są w sterowaniu podsłownikami. Pierwsza wskazuje podsłownik, od którego rozpoczynają się poszukiwania, druga ten, do którego w danej chwili wprowadzane są nowe definicje. Zazwyczaj obie zmienne wskazują na ten sam podsłownik. Istnieje słowo DEFINITIONS, które powoduje, że podsłownik otrzymuje pierwszeństwo w wykonywaniu jak i definiowaniu słów. Pisząc

```
EDITOR DEFINITIONS
```

czynimy słownik edytora słownikiem current i context. W obu wypadkach będzie on brany pod uwagę jako pierwszy.
Do tworzenie, nowych słowników służy VOCABULARY. Pisząc, na przykład

```
VOCABULARY PODWOJNE  PODWOJNE DEFINITIONS
```

możemy najpierw stworzyć słownik PODWOJNE, a potem umieszczać w nim, powiedzmy, zestaw dodatkowych słów do przetwarzania. liczb podwójnej długości. Oczywiście, w ten sposób możemy wyodrębnić jakikolwiek słownik specjalistyczny, którego stworzenie uznamy za celowe W obrębie podsłownika możemy z kolei tworzyć następne piętra podsłowników.

Z pomocą FORGET można skasować grupy słów, jak i całe podsłowniki. Należy jednak pamiętać, że zostaną przy tym skasowane wszystkie słowa zdefiniowane później bez względu na to, do jakiego wprowadziliśmy je podsłownika.

Adres, pod którym dołączany jest nowy podsłownik, podaje w. wielu systemach zmienna VOC-LINK.

Mechanizm sterowania podsłownikami może być różny. W implementacji fig-FORTH przedstawia się on następująco. Podsłowniki osobnym systemem łączników powiązane są w postaci drzewa grafowego, którego korzeniem jest nazwa podstawowego podsłownika FORTH. Tak więc jeżeli przy przeszukiwaniu podsłownika nie zostanie znalezione słowo o podanej przez nas nazwie, poszukiwania zawsze dojdą do podsłownika FORTH. Ten mechanizm powiązań tworzony jest z pomocą słowa VOCABULARY, które ma specyficzną cechę: należy do rodziny słów zdolnych tworzyć nowe słowa. Do definicji VOCABULARY powrócimy w punkcie 8.4.

8.2 Słowa natychmiastowe

W części I ogólnie omówiliśmy tryb wprowadzania do słownika czyli kompilowania nowych słów. Teraz omówimy szczegółowiej mechanizm kompilacji.

Ktokolwiek uważniej przyjrzał się budowie słów FORTH-a w pamięci z pomocą zdefiniowanego przez nas DUMP, mógł dostrzec, że niektóre adresy w polu parametrów prowadzą do słów, których jeszcze nie poznaliśmy, a w dodatku takich, których w ogóle nie użyliśmy w definicji, jak BRANCH, OBRANCH, BACK, LIT i in. Dzieje się tak dlatego, że FORTH rozporządza dodatkowymi słowami służącymi do wykonania kompilacji słowa. Mechanizm definiowania słów, jak zresztą również przebiegania wzdłuż "kodu zszywanego" podczas ich wykonywania oparty jest na wykorzystaniu dwóch zmiennych: wskaźnika instrukcji IP i wskaźnika pracy W. Pierwszy wskazuje adres następnej instrukcji do wykonania, a drugi tej, która jest wykonywana w danej chwili. Z pomocą tych obu wskaźników interpretator ustala kolejność kompilowania fragmentów słowa.

W przypadku, gdy do słowa ma być wkompilowana liczba, interpretator poprzedza ją w definicji słowem, a ściślej adresem jego pola kodu, LIT lub dla liczb jednobajtowych CLIT. największe zmiany w stosunku do tego, co napiszemy z klawiatury, zachodzą przy kompilowaniu konstrukcji warunkowych oraz pętli liczonych i warunkowych. Wynika to z definicji słów budujących takie struktury. Należą one do niezbyt licznych, lecz ważnych słów natychmiastowych (immediate words) .

Na czym polega specyfika słów natychmiastowych? Na tym przede wszystkim, że głównym, w szeregu wypadków jedynym, ich zastosowaniem jest działanie we wnętrzu definiowanych przez nas słów, właśnie działanie natychmiastowe, jak o tym mówi ich nazwa. Przekształcenie słowa w natychmiastowe wykonuje umieszczone bezpośrednio za nim słowo IMMEDIATE. Również jego adresu nie znajdziemy w polu parametrów skompilowanej definicji. IMMEDIATE wykonuje mianowicie jedną czynność; ustawia na 1 bit nr 6 w bajcie długości nazwy słowa, bit P. Daje tym samym znać interpretatorowi, że chodzi o słowo natychmiastowe.

Najłatwiej jest odróżnić słowa natychmiastowe od innych wyświetlając pamięć w hex. Wówczas bajt długości słowa nienatychmiastowego będzie zawierał liczbę 80 + długość nazwy. Natomiast słowo natychmiastowe będzie miało w tym bajcie liczbę dwucyfrową o postaci C + dł. nazwy. Tak więc po pierwszej cyfrze odróżnimy te dwa typy słów.

Definicja IMMEDIATE jest zwykle następująca:

```
: IMMEDIATE LATEST CLIT 64 TOGGLE ;
```

Poznajemy tu słowo TOGGLE, które dodaje do liczby wzór bitowy liczby umieszczonej na szczycie stosu. W tym wypadku do bajtu długości nazwy ostatniego słowa w słowniku, które właśnie definiujemy, TOGGLE dodaje odpowiednik 64, czyli ustawia na 1 bit nr 6 w bajcie nazwy tego słowa.

W definicjach, które tworzone są z użyciem słów natychmiastowych, pojawia się zazwyczaj inne ważne w tej fazie słowo: COMPILE. Powoduje ono, że adres słowa po nim następującego zostaje wkompilowany do tworzonej definicji. COMPILE definiowane jest zwykle następująco:

```
: DO COMPILE (DO) HERE 3 ; IMMEDIATE

: LOOP 3 ?PAIRS COMPILE (LOOP) BACK ; IMMEDIATE
```

Użyte w definicji ?COMP sprawdza, czy system znajduje się w fazie kompilacji, bowiem tylko w niej działa COMPILE. Wgłębiając się w budowę słów natychmiastowych przytoczmy typowe definicje DO i LOOP.

Jak widać, obie definicje wkompilowują adresy odpowiednio (DO) i (LOOP) . W LOOP znajdujemy BACK, którego działanie odpowiada znaczeniu nazwy: przenosi ono akcję wstecz do DO. Jest tu ponadto zastosowany ciekawy chwyt: definicje DO na końcu, a LOOP na początku wprowadzają na stos 3. ?PAIRS bada, czy jest owych trójek para. Ta prosta metoda zapewnia kontrolę, czy istnieją odpowiednie pary DO i LOOP. Gdy tak nie jest, pojawia się komunikat o błędzie.

Podobny mechanizm działa w konstrukcjach IF...THEN i IF...ELSE...THEN.

```
: IF COMPILE 0BRANCH HERE 0 , 2  ; IMMEDIATE

: ELSE 2 ?PAIRS COMPILE BRANCH HERE 0

     , SWAP 2 [COMPILE] ENDIF 2 ; IMMEDIATE

: ENDIF ?COMP 2 ?PAIRS HERE OVER - SWAP

          ! ; IMMEDIATE
```

Szczegółową analizę tych definicji pozostawiamy Czytelnikom. Zwróćmy jedynie uwagę, że również tu zastosowano badanie odpowiedniości słów, tyle że tym razem do kontroli użyta jest liczba 2. W konstrukcjach zaczynających się od BEGIN używa się jedynki. Tym samym zapobiega się również mylnemu kojarzeniu słów, np. niepoprawnemu BEGIN...LOOP, co zostanie niezwłocznie wykryte. W definicji ELSE użyte zostało słowo [COMPILE] . Ma ono działanie odwrotne w stosunku do COMPILE, tzn. wymusza kompilację słowa natychmiastowego.

8.3 Słowa tworzące rodziny słów

Doszliśmy do jednej z najważniejszych właściwości FORTH-a: do jego zdolności nieograniczonego rozbudowywania coraz to nowych struktur danych, plastycznego dostosowywania ich do potrzeb, tworzenia słów wyższego poziomu, z których pomocą można definiować słowa z niższego jakby piętra.

Podstawową konstrukcją, która do tego służy jest BUILDS...DOES . Takie jest brzmienie tej pary słów w fig-FORTH. W niektórych innych wersjach języka pierwsze z nich

zastępuje się słowem CREATE. Znajduje się ono również w słowniku fig-FORTH-a, lecz pełni inne funkcje, a mianowicie umożliwia tworzenie nowego wejścia słownikowego.

Podstawowe znaczenie omawianej tu konstrukcji polega na tym, że zawierające ją słowo określa typ danych, który z jego pomocą tworzymy, i umożliwia definiowanie nieograniczonej liczby słów tego typu, Na przykład, możliwe jest zdefiniowanie rodziny tablic jednowymiarowych lub dowolnego innego wymiaru, łańcuchów znakowych itd. Możemy także tworzyć nowe struktury kontrolne, jeżeli uznamy za niewystarczający ich zestaw istniejący w FORTH.

Powstaje zatem nieograniczona możliwość definiowania nowych słów definiujących czyli takich, jak poznane dotychczas: dwukropek, CONSTANT i VARIABLE.

Składnia omawianej konstrukcji przedstawia się następująco:

```
nazwa typu <BUILDS działanie w fazie kompilacji
            DOES> działania w fazie wykonywania ;
```

Wyjaśnijmy to na stosunkowo prostym przykładzie słowa, z którego pomocą zbudujemy słowa podające nam daty rozmaitych historycznych wydarzeń:

```
: DATY <BUILDS  ,  ,  ,   DOES>

        DUP  2+  DUP  2+  ?  ?  ?  CR  ;
```

Powiedzmy, że chcemy podawać daty w kolejności: dzień, miesiąc, rok. Część za <BUILDS spowoduje, że trzy podane przez nas składniki daty znajdą się na szczycie stosu.
Część znajdująca się na DOES> spowoduje, że spod kolejnych adresów położonych w odstępie dwóch bajtów w słowach, definiowanych z pomocą DATY drukowane będą ich zawartości.
Definiowanie nowych słów z pomocą słowa DATY odbywa się następująco (tu definiujemy daty bitew) :

```
12 9 1683 DATY WIEDEN

 4 4 1794 DATY RACLAWICE

18 6 1815 DATY WATERLOO
```

```
 1  9 1870 DATY SEDAN
12 10 1943 DATY LENINO
```

A oto wywołanie jednego z tych słów:

```
WATERLOO 18 6 1815
ok.
```

Konstrukcja, o której mówimy, stanowi podstawową treść definicji wspomnianego już słowa VOCABULARY, które służy do sterowania mechanizmem podsłowników. Tak więc i tu mamy do czynienia ze słowem definiującym całą rodzinę pokrewnych słów, stanowiących nagłówki poszczególnych podsłowników. Uwidacznia się to w definicji VOCABULARY:

```
: VOCABULARY  <BUILDS  A081
CURRENT  @  2  -  ,
HERE  VOC-LINK  @ ,
    VOC-LINK  !  DOES>
2+  CONTEXT  !  ;
```

Parametr wprowadzany przez pierwszy przecinek - to liczba, podana tu dla jasności w hex, stanowiąca równoważnik pola nazwy słowa "spacja". Jak wiemy, liczby dwubajtowe mają w pamięci komputera odwróconą kolejność bajtów. Tak więc pierwszy bajt, 81 hex, będzie oznaczał bajt długości słowa jednoznakowego, a AO, które zajmie następny bajt - to heksadecymalny odpowiednik 32 - kodu spacji, powiększony o 128, ponieważ FORTH zaznacza w ten sposób ostatni znak nazwy, który tu jest znakiem jedynym.

Drugi parametr, wprowadzany przecinkiem będzie stanowić początek ciągu pól łącznika wszystkich słów, które zostaną zdefiniowane w tym słowniku. Parametr ten zaznacza najpierw początkowy punkt podsłownika, po którym nowy podsłownik zostanie zdefiniowany. W tym wypadku jest to podsłownik podstawowy FORTH. Zapewnia to, że nowy podsłownik będzie rósł jako odgałęzienie starego. Część definicji za DOES> zapewnia, że adres nowego podsłownika zostanie wprowadzony do zmiennej CON-TEXT, gdy podsłownik ten zostanie wywołany.

Wreszcie trzeci parametr tworzy część łańcucha łączącego wszystkie podsłowniki. Tak więc słowo VOCABULARY zapewnia, że tworzone z jego pomocą nagłówki podsłowników łączą je wszystkie w jedną całość zapewniając zarazem autonomiczny charakter każdego podsłownika. Jest to rozwiązanie bardzo interesującej z teoretycznego i praktycznego punktu widzenia.

Konstrukcja <BUILDS...DOES> znajduje szerokie zastosowanie w bardziej zaawansowanym programowaniu w FORTH. Nieraz jeszcze w tej książce spotkamy się z przykładami jej skutecznego użycia.

8.4 Definiowanie w kodzie maszynowym

W FORTH, wspominaliśmy o tym, możliwe jest wprowadzenie do słownika słów zdefiniowanych w kodzie maszynowym częściowo lub w całości. W implementacji fig-FORTH-a słowo ;CODE pozwala dołączyć do definiowanego słowa fragment w języku maszynowym. W taki właśnie sposób definiowane są cztery poznane już przez nas słowa definiujące:

```
: CONSTANT CREATE SMUDGE , ;CODE kod dla stałych

: VARIABLE 0 CONSTANT ;CODE kod dla zmiennych

: USER CONSTANT ;CODE kod dla zmiennych użytkownika

:  :  ?EXEC  !CSP CURRENT @ CONTEXT ! CREATE ]

; CODE kod dwukropka IMMEDIATE
```

W tej ostatniej definicji użyty został końcowy nawias kwadratowy. Powoduje on wyjście ze stanu natychmiastowej kompilacji. Natomiast zamykając fragment definicji w nawiasy kwadratowe powodujemy, że zostaje on wykonany natychmiast już podczas tworzenia definicji. Możemy np. w obrębie nawiasów kwadratowych przewidzieć wykonanie pewnych obliczeń, których w y n i k zostanie umieszczony w definicji. Możemy także zamykając słowo FORTH-a w nawiasy kwadratowe spowodować, że stanie się ono słowem natychmiastowym. Przykłady tego spotykamy dalej.

Istnieje ponadto możliwość definiowania słów wyłącznie w kodzie maszynowym. Służy do tego w wielu implementacjach,

w tym w fig-FORTH, słowo CODE. Tworzy ono nowe wejście słownikowe, a w jego polu kodu umieszcza po prostu adres wskazujący pole parametrów nowego słowa zbudowanego całkowicie w kodzie maszynowym - oczywiście przy zastosowaniu asemblera FORTH-a.

Na zakończenie tego krótkiego przeglądu metod definiowania wymieńmy dwa użyteczne w tej fazie słowa. ID. drukuje nazwę wejścia słownikowego po podaniu adresu pola nazwy. -FIND pozwala ustalić, czy słowo tekstu odgraniczone spacjami, które umieścimy przed -FIND, jest już nazwą obecną w słowniku. W takim wypadku pojawia się na stosie adres pola parametrów tego słowa, jego bajt długości i znacznik "prawda". W przeciwnym - znacznik "fałsz".
Mechanizmy kompilacji, które omówiliśmy dotychczas w tym rozdziale, zapewniają zróżnicowane formy definiowania słów.

8.5 Rekurencja

Rekurencja - to pojęcie i obszar zagadnień mających bez wątpienia nieporównanie szerszy charakter niż temat tego rozdziału poświęconego problemom kompilacji w FORTH.
Teoria rekurencji - to wielki i ważny dział logiki matematycznej i podstaw matematyki. Z nich też pojęcie i metoda przeniosły się do informatyki i oznaczają w niej, najogólniej mówiąc, postępowanie polegające na wywołaniu funkcji z jej wnętrza.

W sporach o kierunek rozwoju informatyki ścierały się nieraz przeciwstawne tendencje. Dijkstra [11, s. XII] nie bez ironii zauważa, że co bardziej teoretyzujący informatycy uważają programy rekurencyjne za "bardziej naturalne" niż iteracyjne. Sam Dijkstra swoją znakomitą "Umiejętność programowania" zbudował pomijając rekurencję. Praktyka dowiodła, że procedury rekurencyjne można na ogół przekształcić w iteracyjne, czyli oparte na wykorzystaniu pętli. Procedury rekurencyjne dają się na ogół wyrazić przejrzyściej i zwięźlej niż iteracyjne, natomiast ich wykonanie zabiera więcej czasu i miejsca w pamięci.

Czasem trudno jest dla procedury rekurencyjnej znaleźć jej iteracyjny odpowiednik, a do takiego typu problemów należy zwłaszcza klasyczny w matematyce i informatyce problem wież w Hanoi. Jego rekurencyjną postać w FORTH przedstawiamy w rozdz. 12.

Zdawałoby się, że w FORTH wywołanie funkcji z jej wnętrza nie jest w ogóle możliwe, ponieważ w słowie nie można użyć nazwy słowa jeszcze nie zdefiniowanego. Znaleziono jednak pomysłowe rozwiązania pozwalające przezwyciężyć te trudności. Pierwsza droga ominięcia ograniczeń polega na stworzeniu słowa natychmiastowego, które w czasie kompilowania definiowanego słowa umieści w jego polu parametrów w miejscu odpowiadającym wywołaniu tego słowa adres owego pola parametrów. Oto takie słowo:

```
: TENKOD CURRENT @ @ PFA CFA ; IMMEDIATE
```

Teraz wywołanie słowa z jego wnętrza zapewni sekwencja trzech słów: TENKOD LITERAL EXECUTE. Zastosujmy tę metodę do obliczania kolejnych wartości silni. Oto iteracyjna i rekurencyjna definicja tej procedury:

```
: SILNIA 1 SWAP 1+ 1 DO I * LOOP U. ;

: SILNIA2 DUP 2 = IF ELSE DUP 1 -
        TENKOD LITERAL EXECUTE * THEN ;
```

Istnieje jednak elegantsze rozwiązanie. Napiszmy najpierw definicję, a potem wyjaśnimy mechanizm:

```
: SILNIA3 [ SMUDGE ] DUP 2 = IF ELSE
        DUP 1 - SILNIA3 * THEN [ SMUDGE ] ;
```

Widoczne jest duże podobieństwo dwu ostatnich definicji. Różnica polega na zastąpieniu poprzedniej triady słów napisaną wprost nazwą procedury wywoływanej z jej wnętrza oraz na pojawieniu się na początku i końcu definicji słowa SMUDGE ujętego w nawiasy kwadratowe. Dlaczego mimo wpisania do słowa jego własnej nazwy nie został zasygnalizowany błąd? Dzie-

je się tak za sprawą pierwszego SMUDGE. Ustawia ono mianowicie szósty od prawej bit w bajcie nazwy na zero (w innych systemach na 1) symulując tym samym, że słowo jest już zdefiniowane. Ponieważ czynność ta musi być wykonana w toku kompilowania definicji czyli natychmiast, a SMUDGE w większości implementacji nie jest słowem natychmiastowym, trzeba mu nadać taki charakter, a potem pozbawić je go. Czyni to para słów: otwierający i zamykający nawias kwadratowy. Pamiętajmy, że są to słowa i muszą być od sąsiednich słów lub wartości oddzielone spacjami.

Dlaczego na końcu używamy raz jeszcze identycznej sekwencji? Jest to niezbędne, ponieważ czynność ustawienia na 0 bitu znamienia wykonuje również końcowy cudzysłów. Dwukrotne ustawienie bitu sprawiłoby, że stałby się on znowu równy zeru. Stąd potrzeba trzeciej zmiany jego wartości. Wyjaśnienie to jest, być może, bardziej zawiłe, niż sama konstrukcja umożliwiająca stosunkowo łatwe stosowanie rekurencji w FORTH. Należy mieć na uwadze, że omówione rozwiązania są stosowaniem rekurencji w języku, który nie jest na to nastawiony, toteż przy najmniejszej niedokładności przedstawione mechanizmy będą zawodzić.

W niektórych implementacjach FORTH-a stworzono specjalne instrumenty do stosowania rekurencji, np. słowo RECURSE albo też odrębną parę słów R: i R; . FORTH na Jupiter Ace dopuszcza stosowanie rekurencji bez dodatkowych środków.

8.6 Wykonywanie

Definiujemy słowa, by mogły być potem wykonane. Przechodząc w fazę wykonywania interpretator FORTH-a posługuje się szeregiem słów, w tym przede wszystkim kluczowym słowem EXECUTE, co po angielsku znaczy właśnie: wykonaj. Działa ono odwrotnie niż COMPILE, powoduje mianowicie wykonanie słowa, którego adres pola kodu znajduje się na szczycie stosu. Jak pamiętamy, taki właśnie efekt daje napisanie słowa i naciśnięcie Returnu. Interpretator nieustannie wywołuje słowo EXECUTE. Szczegóły działania tego mechanizmu mogą być zależne od typu komputera.

Czynności związane z analizą tekstu źródłowego wpływającego z klawiatury lub urządzenia zewnętrznego skupione są w słowie INTERPRET. Jego rozbudowaną definicję przedstawiamy, bowiem pozwala ona lepiej poznać działanie FORTH-a w fazie wykonywania, jak i w fazie kompilowania nowych słów.

```
: INTERPRET BEGIN BL WORD FIND -DUP IF STATE +
                    IF EXECUTE ELSE , THEN
            ?STACK ELSE NUMBER DPL @ i+
                    IF DLITERAL ELSE DROP LITERAL THEN
            ?STACK THEN
    UNTIL  ;
```

Po napisaniu słowa lub przy odczytywaniu go z urządzenia zewnętrznego WORD wczytuje je na szczyt słownika, FIND sprawdza, czy słowo znajduje się w słowniku. Gdy go nie ma, na stosie pojawia się zero. W przeciwnym razie interpretacja postępuje dalej naprzód. Jeżeli słowo jest natychmiastowe, zostaje niezwłocznie wykonane. W przeciwnym wypadku następuje sięgnięcie do jego kodu zszywanego. W przypadku nie znalezienia słowa w słowniku INTERPRET podejmuje próbę odczytania tego, co napisaliśmy, jako liczby, co wykonuje słowo NUMBER. W definicji znajdują się również słowa LITERAL i DLITERAL, których specyfika polega na tym, że działają tylko w fazie kompilacji.
Działanie LITERAL wyjaśnijmy z pomocą przykładu. Powiedzmy, że w czasie opracowywania definicji doszliśmy do wniosku, że celowe byłoby wprowadzenie do niej wyniku pomocniczego obliczenia. Jeżeli napiszemy definicję

```
: LICZBA [ 2 7 * 6  + 5/ ] LITERAL ;
```

to okaże się, że słowo LICZBA działa tak jak stała i podaje na stos wartość stanowiącą wynik obliczenia ujętego w nawiasy kwadratowe. Obliczenie to zostanie wykonane w czasie definiowania słowa i z pomocą LITERAL jego wynik zostanie wprowadzony do definicji. W rzeczywistości słowo LICZBA będzie miało w słowniku bardzo prostą konstrukcję, którą można wyrazić następująco:

```
: LICZBA LIT 4 ;
```

4 - to wynik obliczenia wyrażenia zawartego w nawiasach kwadratowych. Ten prosty przykład ukazuje znacznie szersze dostępne w FORTH możliwości wprowadzania do słownika takich słów, w których duża część obliczeń, zwykle znacznie bardziej zawiłych niż w podanym przykładzie, zostanie wykonana już w fazie kompilacji. Nie trzeba podkreślać, jak bardzo w niejednym przypadku skrócić to może czas wykonania.

Kontrolę, czy tekst wpływający z klawiatury bądź z urządzenia zewnętrznego ma charakter słowa definiowanego czy też przeznaczonego do natychmiastowego wykonania, zapewnia zmienna STATE. Przybiera ona wartości niezerowe, gdy system znajduje się w fazie kompilacji. Tylko w takim przypadku działają DLITERAL i LITERAL powodujące wywołanie LIT i skompilowania napotkanej liczby lub wyrażenia w definicji tworzonego słowa. Tak więc słowo INTERPRET jest w systemie FORTH-a swego rodzaju górką rozrządową wyznaczającą różne tory postępowania stosownie do charakteru tekstu źródłowego poddawanego interpretacji. Działa ono w pętli, dopóki "ma co robić".

Na marginesie słów, które ostatnio poznaliśmy, a w szczególności COMPILE, EXECUTE i INTERPRET, a także LITERAL i DLITERAL warto zauważyć, że słowa te w bardzo szerokim stopniu są wykorzystywane przez interpretator, są w istocie jego częściami składowymi. Użytkownik korzysta z nich na ogół rzadziej i przy podejmowaniu bardziej złożonych zadań programistycznych.

Uwaga ta przeznaczona jest zwłaszcza dla tych, których niepokoi duża liczba predefiniowanych słów FORTH-a. Nie, trzeba się nią przejmować. Stopniowo, w miarę poznawania języka nawet początkujący programista dojdzie do chwili, w której odsłoni się przed nim całościowa konstrukcja FORTH-a, A wtedy zobaczy, że każde słowo predefiniowane ma w niej jakąś rolę do spełnienia, że czasem jest to rola mała i cząstkowa, a kiedy indziej znaczna, lecz po części ukryta przed naszym okiem, jak to się dzieje w przypadku omówionych ostatnio słów.

Jeżeli mimo upływu kilkunastu lat, czyli w rozwoju informatyki całej epoki, FORTH jest nadal językiem żywym, a

krąg jego użytkowników rozszerza się, zawdzięcza to głównie temu, że jest w rękach posługującego się nim programisty narzędziem nad wyraz sprawnym, pozwalającym od czynności podstawowych i najprostszych dochodzić do podejmowania zadań wymagających najwyższych talentów twórczych.

8.7 Kontrola błędów

W FORTH działają dwa mechanizmy kontroli błędów. Jeden polega na tym, że gdy interpretator w fazie wykonywania stwierdzi błąd, poszukuje jego określenia w jednym z dwóch ekranów zawierających komunikaty o błędach, po czym określenie to drukuje. Drugi mechanizm polega na wykorzystaniu przez użytkownika w fazie kompilacji pewnych specjalnych słów zapewniających kontrolę błędów. Niektóre z tych słów, jak ?COMP i ?PAIRS, poznaliśmy wcześniej.

Omówiona przed chwilą definicja INTERPRET zawiera słowo ?STACK kontrolujące, czy nie nastąpiło przepełnienie lub opróżnienie stosu. Inne słowo, !CSP, wprowadza aktualny wskaźnik stosu do zmiennej CSP, a ?CSP uruchamia procesor błędów, gdy aktualny wskaźnik stosu SP nie jest równy wartości CSP. Jest to cenny instrument kontroli przy uruchamianiu programów.
Najczęściej stosowane formy kontroli błędów opierają się na słowie ?ERROR, a to z kolei na słowie ERROR. Na przykład:

```
: ?PAIRS - 19 ?ERROR ;

: ?ERROR SWAP IF ERROR ELSE DROP THEN ;
```

Jeżeli ?ERROR nie napotka zera, wywołuje ERROR, a to słowo z kolei przy pomocy MESSAGE powoduje wyświetlenie komunikatu. W danym wypadku jest to komunikat 19 czyli "Conditionals not paired". Inne słowa kontrolujące błędy powodują także ewentualne wyświetlenie odpowiednich komunikatów o błędach. Zwyczajowo zaczynają się one od znaku zapytania.

Słowo MESSAGE może być użyteczne także do innego celu w przypadku posługiwania się stacją dyskietek. Na drugim ekranie z komunikatami błędów po ostatnim z nich zajmującym 24 pozycje są jeszcze wolne linie. Można tam wpisać dowolny komunikat, na przykład, stały nagłówek stron dla drukarki i wywołując

MESSAGE z odpowiednim numerem linii wyższym od 24 powodować druk tego komunikatu. Numerację linii dla MESSAGE można dowolnie zwiększać, jeżeli na następnych ekranach dyskietki chce się umieścić dalsze napisy. Kontrolę tego mechanizmu wykonuje zmienna WARNING.

8.8 Inicjalizacja

W sytuacjach awaryjnych, lecz także do sterowania przebiegiem programu można wykorzystać trzy słowa dokonujące w nieco odmienny sposób inicjalizacji FORTH-a, czyli przywrócenia początkowego stanu jego pracy.

QUIT opróżnia jedynie stos powrotów.

ABORT opróżnia oba stosy, a ponadto wyświetla początkowy nagłówek interpretatora.

COLD ma działanie najdalej idące, bowiem dodatkowo oczyszcza bufory wejścia/wyjścia i przywraca stan początkowy zmiennym użytkownika. Procedura COLD jest wywoływana za każdym razem, gdy interpretator FORTH-a zaczyna działać po jego wgraniu do pamięci. Nazywane to jest zimnym startem systemu.

ROZDZIAŁ 9
LICZBY RAZ JESZCZE

9.1 Tablice

Wśród wielu celów, do jakich można zastosować konstrukcję <BUILDS...DOES>, jednym z istotnych jest tworzenie instrumentów zapewniających możliwość posługiwania się tablicami różnego rodzaju: od łańcuchów znakowych, przez tablice jednowymiarowe, do tablic o dwu lub więcej wymiarach.

Tablicę jednowymiarową definiujemy następująco:

```
: ARRAY <BUILDS 2 * ALLOT DOES> SWAP 2 * +  ;
```

Słowo ARRAY umożliwia odtąd definiowanie dowolnej liczby tablic jednowymiarowych o różnych długościach. Czynimy to w następujący sposób:

```
10 ARRAY LICZBY
```

Zdefiniowaliśmy tablicę o nazwie LICZBY pozwalającą umieścić w niej dziesięć liczb pojedynczej długości. Część definicji ARRAY znajdująca się za <BUILDS powoduje, że przy definiowaniu słowa LICZBY zostaje na nie zarezerwowanych 20 bajtów. Druga część definicji ARRAY określa sposób, jednolity dla wszystkich ustanawianych z jej pomocą tablic, w jaki uzyskiwać będziemy dostęp do tablicy, w celu umieszczania w niej, pobierania i zmiany wartości. Pisząc teraz

```
7 LICZBY
```

powodujemy, że na stos wprowadzony zostaje adres siódmej komórki dwubajtowej w tablicy LICZBY. Jeżeli chcemy do tej komórki wstawić liczbę, powiedzmy, 22, piszemy:

```
22 7 LICZBY !
```

Podobnie odczytujemy wartość:

```
7 LICZBY @ . /Return/ 22 ok
```

Tablica w ten sposób zdefiniowana zawiera komórki o adresach od 0 do 9.

Sposób definiowania tablicy dwuwymiarowej jest nieco bardziej skomplikowany. Oto słowo zaczerpnięte z [6] .

```
: 2DARRAY <BUILDS

      DUP ,   ( wpisuje wymiar Dim2   )

* 2 * ALLOT  ( rezerwuje miejsce   )

DOES>

     ROT      ( przenosi wskaźnik i1 na szczyt stosu)

OVER @ +      ( mnoży i1 przez Dim2 )

ROT +         ( dodaje wskaźnik i2 )

2 * + 2+ ;  ( oblicza adres komórki (i1,i2) )
```

Przez i1 i i2 oznaczone zostały wskaźniki tablicy dwuwymiarowej. Ponieważ tablica dwuwymiarowa stanowi w pamięci komputera jednowymiarowy ciąg komórek, pierwsza część definicji 2DARRAY rezerwować będzie w nowo definiowanych tablicach miejsce na podstawie przemnożenia podanych wymiarów tablicy oraz podwojenia tej wielkości, ponieważ każda liczba zajmuje dwa bajty. Wpisany ponadto zostaje drugi wymiar tablicy.

Druga część definicji 2DARRAY przeznaczona jest na fazę wykonywania, czyli uzyskiwania dostępu do komórek tablicy, którą zdefiniujemy. Końcowe 2+ powoduje, że adres powiększony zostaje o 2, tak iż komórka zerowa staje się pierwszą, pierwszą drugą itd. Pisząc

```
4 5 2DARRAY PROSTOKAT
```

definiujemy tablicę prostokąt o elementach numerowanych od 0,0 do 3,4.

W literaturze znaleźć można również inne mechanizmy definiowania tablic, dwuwymiarowych. Oto np. w [5] podaje się następujące słowo:

```
:  MATDEF ( dim1 dim2 --- )
        <BUILDS
        OVER  OVER  SWAP  ,  ,  *  2  *  ALLOT
                DOES>                ( i1  i2 pfa+2 )
        >R                           ( i1  i2 )
        OVER  OVER                   ( i1 12 i1 12 )
        R  @                         ( i1 i2 i1 i2 dim1 )
        R  2  +  @                   ( i1 12 i1 i2 dim1 dim2 )
        ROT  -  0>  ROT
        ROT  -  0<                   ( ind1  ind2 f1 f2 )
                *                    ( ind1 ind2 f )
IF  R@  *  +  2  *  R>  4  +         ( przesunięcie pfa+6 )
        +                            ( adres )
ELSE  R>  DROP  DROP  DROP
                ." ZLE WSKAZNIKI"    ( sygnał o błędzie )
THEN ;
```

W definicji tej, podobnej w zasadniczej strukturze do poprzedniej, zwraca uwagę zastosowanie kontroli, czy którykolwiek z wprowadzanych wskaźników nie powoduje przekroczenia wymiarów tablicy. Jest to istotne zabezpieczenie przed omyłkowym wstawieniem wartości w niewłaściwe miejsce w słowniku, co może powodować uszkodzenia w słowach, które się tam znajdują.

Ciekawym rozwiązaniem jest słowo, które wyświetlać będzie zawartość każdej tablicy zdefiniowanej z pomocą MATDEF. Należy stosować składnię: MATDISP nazwa-tablicy.

```
        : MATDISP  [COMPILE]' 6 + DUP >R 4 - @ R>

                2 - @  0

        DO SWAP OVER 0 DO DUP ? 2+ LOOP

                SWAP LOOP DROP DROP  ;
```

I jeszcze jedna definicja:

```
:  ARRAY2
        <BUILDS
        OVER  ,  *  ALLOT
        DOES>
        DUP  @  ROT  *  +  +  2+  ;
```

Tablice dwuwymiarowe definiowane z pomocą ARRAY2 przeznaczone są do liczb jednobajtowych. Tak oto definiujemy MATRIX/2,3/:

```
2  3  ARRAY2  MATRIX
```

9.2 Formatowanie liczb

Niektórzy, zwłaszcza mniej doświadczeni, programiści mogą uznać za szczególnie kłopotliwe to, że FORTH posługuje się liczbami całkowitymi, gdy niejednokrotnie zachodzi potrzeba wykonywania działań na ułamkach dziesiętnych. Otóż w FORTH istnieją co najmniej dwa sposoby przezwyciężenia tej trudności.

Zanim je omówimy, warto poczynić parę uwag ogólniejszej natury. W rzeczywistości komputer wykonuje wszystkie obliczenia na liczbach całkowitych i to w dwójkowym układzie pozycyjnym. Następną cechą arytmetyki komputerowej jest to, że działania na liczbach, takie jak dodawanie, odejmowanie lub mnożenie, realizowane są z pomocą podprogramów, których złożoność wzrasta ze zwiększeniem się długości liczb. Im więcej bajtów zajmują liczby, tym bardziej skomplikowane jest wykonywanie działań na nich, z czym nieuchronnie wiąże się wydłużenie czasu obliczeń. Szczególnie trudnym orzechem do zgryzienia jest dzielenie, co jest skądinąd zgodne z naszymi doświadczeniami. Przyczyna, mówiąc w uproszczeniu, tkwi w tym, że dzielimy liczby "od przodu", gdy pozostałe trzy działania wykonujemy "od tyłu".

I wreszcie ostatnia i najważniejsza właściwość: to, co oglądamy na ekranie lub wydruku jako liczby i wyniki działań, jest w rzeczywistości ich odwzorowaniem w postaci łańcuchów znaków, którymi są znaki cyfr dopuszczalnych w zastosowanym w

danej chwili układzie liczbowym. W układzie dziesiętnym będą to znaki cyfr od 0 do 9. Ponadto w takim wyprowadzanym łańcuchu może się znajdować kropka oddzielająca część całkowitą od ułamkowej. Ustalenie miejsca tej kropki następuje w wyniku wykonania specjalnego podprogramu.

Podstawowym sposobem przetwarzania liczb mających część ułamkową jest przejście na arytmetykę zmiennopozycyjną. Metoda ta omówiona będzie w następnym punkcie.

Tu zajmiemy się inną metodą, która ma tę zaletę, że nie wymaga przekroczenia granicy liczb podwójnej długości, a umożliwia przedstawianie liczb z umieszczoną na właściwym miejscu kropką zamykającą część całkowitą.

Jest to formatowane wyprowadzanie liczb.

Zacznijmy od omówienia mechanizmu wyprowadzania liczb podwójnych bez znaku, ponieważ dla procesu formatowania jest to zakres podstawowy. Inne rodzaje liczb wymagają właściwego ich przygotowania do formatowania.

Początek formatowania wydruku zaznaczony jest słowem <# . Inicjuje ono specjalny bufor znaków poniżej PAD. Definicja tego słowa brzmi:

```
: <#  PAD  HLD  !  ;
```

Zmienna HLD zapamiętuje adres PAD.

ten znak noszący nazwę "hasz" jest słowem powodującym obliczenie i wyprowadzenie pierwszej z prawej, czyli najmniej znaczącej cyfry liczby. Wyprowadza ją odpowiednio do obowiązującej podstawy układu liczbowego, np. dziesiątkowego lub szesnastkowego. Z pomocą tego słowa wyprowadzamy również następne cyfry. Jeżeli trzykrotnie użyjemy znaku hasz, wyprowadzone zostaną trzy kolejne cyfry od końca, jeżeli liczba nie ma ich mniej.

Z pomocą hasza wyprowadzamy zwykle cyfry do kropki odgraniczającej część ułamkową. Wstawienia kropki do wyprowadzanego łańcucha dokonujemy z pomocą słowa HOLD. Może ono wstawić dowolny znak, bowiem działa podobnie jak EMIT, tyle że w obrębie formatowania liczb. Resztę konwersji liczby możemy wykonać z pomocą jednego już tylko słowa #S. Wykonuje ono konwersję wszystkich pozostałych cyfr znaczących liczby. Jakkolwiek mówimy tu o liczbach bez znaku, dodajmy, że przed zakończeniem formatowania można posłużyć się słowem SIGN. Dopisuje ono znak minusa wtedy, gdy trzecia liczba od szczytu stosu jest liczbą ujemną. Definicja SIGN może być następująca:

```
: SIGN ROT 0< IF 45 HOLD THEN ;
```

45 to kod ASCII znaku minusa. W niektórych systemach zasady działania SIGN mogą być inne.
Ostatnim słowem stosowanym w formatowanym wyprowadzeniu liczb jest #> . Kończy ono omówioną tu sekwencję.
Zdefiniujmy słowo, które będzie wyprowadzać liczby z dwu cyframi za kropką dziesiętną.

```
: SETNE <# # # 46 HOLD #S #> TYPE ;

123489. SETNE Return  1234.89 ok
```

A oto inny prosty przykład zastosowania formatowanego wyprowadzenia. Tworzymy słono, które liczbę sekund przeliczać będzie
na godziny, minuty i sekundy.

```
: SEXTAL 6 BASE ! ;

: XX # SEXTAL # DECIMAL 58 HOLD ;

: SEK <# XX XX #S #> TYPE ;
```

Liczby inne niż podwójne bez znaku trzeba następująco przygotować do formatowania:

Rodzaj formatowanej liczby	Sposób przygotowania
31-bitowa ze znakiem	SWAP OVER DABS
16-bitowa bez znaku	0
15-bitowa ze znakiem	DUP ABS 0

Przygotowanie takie zakłada, że SIGN w sposób wcześniej opisany będzie zastosowane do liczb podwójnych lub pojedynczych ze znakiem, natomiast nie będzie używane w stosunku do liczb bez znaku. Przypomnijmy, że konwersję w przeciwnym kierunku, tzn. z łańcucha znaków cyfrowych w liczbę podwójną ze znakiem stosownie do aktualnej podstawy zawartej w BASE wykonuje słowo NUMBER.
Formatowanie liczb pozwala bez przechodzenia na arytmetykę zmiennopozycyjną wykonywać działania na ułamkach. Podajemy za [6] definicje dwóch słów umożliwiających dodawanie liczb stałopozycyjnych i wyprowadzanie poprawnych wyników.

```
: FIX DPL @ 0< IF   (jeżeli liczba była pojedyncza)
      S->D 0 DPL !  ( przekształca ją w podwójną )
  THEN
```

```
        DPL  @  DUP  4  <  IF  4  SWAP
        DO  10.  D*  LOOP   ( skaluje )
               ELSE
        4  >  IF  ."  Poza zakresem"  2DROP  THEN  THEN  ;
: F.  SWAP  OVER  DABS
        <#  #  #  #  #  46  HOLD  #S  SIGN  #>  TYPE  SPACE  ;
```

Przykład zastosowania:

```
0.04   FIX ok
0.3    FIX D+ ok
0.625  FIX D+ ok
0.0001 FIX D+ ok
10     FIX D+ ok
F.  10.9651   ok
```

9.3 Jeszcze o liczbach podwójnych

Podstawowy słownik predefiniowany zawiera stosunkowo niewiele operatorów umożliwiających przetwarzanie liczb podwójnej długości. Zdarzają się implementacje zawierające jedynie D+, DABS, DMINUS albo DNEGATE. Skuteczne posługiwanie się liczbami podwójnymi wymaga posiadania specjalnego pakietu słów, które w razie potrzeby mogliśmy wprowadzić do słownika. Nie siląc się na przedstawienie wszystkich możliwych użytecznych słów rozpatrzmy kilka dodatkowych definicji jako uzupełnienie słów omówionych w rozdz.

Szczególnie użyteczne przy posługiwaniu się liczbami przekraczającymi górną granicę podwójnej długości ze znakiem jest słowo UD. drukujące takie liczby.

```
UD.  <#  #S  #>  TYPE  ;
: D-  DMINUS  D+  ;
: D0= OR 0= ;          (  test na liczbę podwójną 0 )
: D0< SWAP DROP 0<  ;  (  test na ujemną liczbę podwójna )
: D=  D- D0= ; ( test  na równość dwóch liczb podwójnych )
: D+-  0<  IF DMINUS  THEN  ;
: DABS  DUP  D+-  ;
```

9.4. Arytmetyka zmiennopozycyjna

W dotychczasowym omawianiu FORTH staraliśmy się wskazać na jego istotną zaletę polegającą na możliwości posługiwania się li-

czbami 16-bitowymi i 32-bitowymi. Omówione środki wskazują, że po nabraniu pewnej wprawy można bez stosowania arytmetyki zmiennopozycyjnej posługiwać się liczbami rzeczywistymi.

Czasem jednak wygodne staje się opuszczenie tradycyjnej domeny FORTH , liczb całkowitych, i przejście na arytmetykę zmiennopozycyjną, taką, jaką posługuje się np. BASIC.

Pakiety do arytmetyki zmiennopozycyjnej stanowią część składową wielu implementacji, np. wspomnianego już Extended fig-FORTH na Atari. Część definicji w takich pakietach napisana jest w asemblerze FORTHA, ponieważ odwołują się one do znajdującego się z reguły w każdym komputerze kompletu podprogramów arytmetyki zmiennopozycyjnej.

OWEN BISHOP [3] przedstawia pomysłową koncepcję zastosowania arytmetyki zmiennopozycyjnej w obrębie tradycyjnego słownika FORTH i nawet bez użycia formatowania liczb. Ma ona jednak szereg ograniczeń, a jej przedstawienie i wyjaśnienie zajęłoby wiele miejsca. Potraktujmy ją jako jeszcze jedno potwierdzenie, że eksperymentowanie w FORTH nie ma granic i w ślad za Bishopem zachęćmy do poszukiwań w tym kierunku.

9.5. Liczby losowe

Skorzystamy z innego pomysłu tego autora: słowa generującego liczby losowe. Są one, jak wiadomo, często potrzebne zarówno w rozmaitych zastosowaniach naukowych, np. przy posługiwaniu się w celach statystycznych próbami losowymi, jak i w grach

Komputery mają podprogramy generujące tzw. liczby pseudolosowe, tzn. nie w pełni odpowiadające kryteriom, Jakie powinny spełniać liczby losowe, ale przydatne do wielu celów. Znając mapę pamięci komputera można takie liczby odczytać pod ustalonym adresem.

Generowanie liczb losowych ma stosunkowo obszerną literaturę, a Peter Naur w "Zarysie metod informatyki"(Warszawa 1979, str. 70-73) omawia jedną z metod. Podobną drogą poszedł Bishop. Najpierw tworzy się zmienną, która będzie zawierać kolejne generowane liczby i umożliwić generowanie następnych:

```
0 VARIABLE LICZBA
```

Następna procedura mnoży poprzednią wartość zmiennej LICZBA przez 2011 i dodaje 5. Obie te liczby są pierwszymi, co jest przy tej operacji zasadą. Wynik działań jest z pomocą działającego na

bitach AND porównywany z 32767. Tak więc RND1 produkuje liczby losowe w granicach od 0 do 32767

```
: RND1 LICZBA @ 2011 * 5 +
32767 AND DUP LICZBA ! ;
```

Teraz z pomocą RND możemy wygenerować liczbę losową w przedziale od 0 do n-1 pisząc n RND.

```
: RND RND1 32767 */ ;
```

Ćwiczenie 11

Posługując się zdefiniowanymi tu słowami stwórzmy słowo, które wygeneruje losowe wyniki stu rzutów kostką do gry.

ROZDZIAŁ 10
AND, OR i XOR

10.1 Operatory logiczne i ich działanie

Mechanizmy obliczeń istniejące w FORTH sprawiają, że tradycyjnie stosowane w wielu językach programowania operatory logiczne AND, OR i NOT mogą być z identycznym skutkiem zastąpione odpowiednio przez *, + i 0= .

Natomiast operatory logiczne istniejące w FORTH mają tę właściwość, że przetwarzają bity dwóch liczb na szczycie słownika i zastępują je wynikiem operacji logicznej. Tak więc działanie operatorów logicznych jest w FORTH takie samo jak w języku maszynowym, co wydatnie rozszerza zakres ich zastosowań.

Podstawowy zestaw słów obejmuje AND, OR i XOR. Zdarzają się również operatory NOR i NAND. Działanie AND, OR i XOR na bitach daje następujące wyniki:

Bit1	Bit2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

AND jest zatem koniunkcją i logicznym odpowiednikiem spójnika "i", OR alternatywą i odpowiednikiem "albo", XOR zaprzeczeniem tożsamości i logicznym odpowiednikiem konstrukcji "albo...albo". Daje Ono wynik "prawda" czyli 1 tylko wtedy, gdy przetwarzane wartości nie są takie same. Albo jedna, albo druga z nich musi być prawdziwa bądź fałszywa, ale gdy są takimi równocześnie, wynik brzmi: "fałsz", Warto zauważyć, że ten najmniej znany z trzech operatorów znajduje szerokie zastosowanie w schematach logicznych maszyny cyfrowej.

Działanie operatorów logicznych na liczbach może się

wydać początkowo niezrozumiałe. Dlaczego, na przykład, 28 14 AND zostawia na stosie 12, 40 10 OR - 42, a 40 10 XOR - 34 ? Jeżeli z pomocą 2 BASE ! przejdziemy na dwójkowy układ liczbowy, przekonamy się, że wyniki te są zgodne z przytoczoną wcześniej matrycą logiczną.

W celu zrobienia prób zdefiniujmy słowo BIN powodujące przejście na układ dwójkowy, wydrukowanie liczby i powrót do układu dziesiątkowego:

```
: BIN 2 BASE ! . DECIMAL ;
```

Napiszmy:

```
28 BIN .Return 11100 ok

14 BIN .Return  1110 ok

28 14 AND BIN . Return 1100 ok
```

W podobny sposób można sprawdzić działanie pozostałych operatorów.
Całkowicie poprawne jest działanie AND i OR, gdy chcemy zinterpretować wynik jako znacznik logiczny w stosunku do całych liczb:

```
nie-zero zero AND daje zero

nie-zero zero OR  daje nie-zero
```

Czy można w liczbie dokonać tak zwanej inwersji bitów, czyli zamienić bity zerowe na jedynkowe i na odwrót? Owszem, tak. Należy zastosować słowo MINUS (NEGATE) , odjąć 1 i wyprowadzić liczbę bez znaku. Można zdefiniować taki operator. Można także posłużyć się XOR, o czym dalej.

10.2 Maski bitowe

Jeżeli napiszemy 77 1 AND, wynik brzmieć będzie 1, a jeżeli napiszemy 78 1 AND, otrzymamy jako wynik zero. Nie jest to przypadek. Każda liczba nieparzysta da wynik 1, a parzysta 0. Tak więc konstrukcja n 1 AND może służyć do sprawdzenia, czy liczba jest nieparzysta.

Dlaczego tak się dzieje? Wyraźniej zobaczymy to, gdy przedstawimy liczby w układzie dwójkowym:

1001101

0000001 AND

0000001

Liczba 1 w układzie dwójkowym poza najmniej znaczącym bitem wszystkie pozostałe ma ustawione na zero. Wszystkie bity zerowe w wyniku operacji AND zastosowanej w stosunku do dowolnych bitów innej liczby dadzą w efekcie 0, bowiem zarówno 1 0 AND, jak i 0 0 AND pozostawiają wynik 0. Mówimy, że zastosowana tu została maska z AND.

Inaczej jest w przypadku OR.

1001101

0100001 OR

1101101

OR spowoduje, że bity liczby, z której pomocą wykonujemy operację OR, zostaną jakby dorzucone do bitów pierwotnej liczby. Mówimy tu również o masce OR. Nie przerzedza ona bitów jedynkowych w liczbie, lecz je jakby zagęszcza dodając nowe tam, gdzie ich nie. było, i nie zmieniając pozostałych jedynek.

Szczególnie skutecznym i bodaj najszerzej, choć często pośrednio, wykorzystywanym operatorem logicznym jest w FORTH XOR. Jego dogodną właściwością jest to, że maska bitowa nałożona z pomocą XOR dokonuje inwersji, czyli odwrócenia wartości bitów wtedy, gdy 1 spotyka się z 1 lub zero z zerem.
Z pomocą XOR możemy łatwo sprawdzić, czy dwie liczby są sobie równe, bowiem wówczas XOR daje wynik zerowy. Przyjrzyjmy się temu na przykładzie dwóch równych liczb przedstawionych w systemie dwójkowym:

110110010101

110110010101 XOR

000000000000

Spotkanie par jednakowych liczb zawsze daje zero. Maska bitowa XOR staje się szczególnie interesująca, gdy operacji dokonujemy z zastosowaniem liczby, w której tylko jeden

bit ma wartość 1. Taką postać binarną mają liczby 2, 4, 8, 16, 32, 64, 128, 256, 512 i wszystkie dalsze potęgi dwóch. Co dzieje się, gdy nałożymy na liczbę taką maskę? Sprawdzi ona tylko jeden bit i jeżeli ma on wartość 1, zamieni go w 0, a gdy ma wartość 0 - zamieni w 1, dokona więc inwersji tylko tego jednego bitu nie zmieniając pozostałych. Na liczbach dziesiętnych znajdzie to następujące odbicie:

```
133  32  XOR  155  ok   155  32  XOR  133  ok
```

Przykład, którym się tu posługujemy, nie jest wyimaginowany. Tak właśnie zachowuje się bit długości nazwy słowa, gdy zmieniamy w nim bit nr 5, czyli bit znamienia, decydujący o tym, czy słowo rozpoznawane jest jako ważne. Taką czynność wykonuje SMUDGE, a przede wszystkim końcowy średnik zamykający wszystkie definicje dwukropkowe. Zarówno jedno, jak i drugie słowo posługuje się w tym celu słowem TOGGLE. To ostatnie jest wprawdzie na ogół zdefiniowane w kodzie maszynowym, ale kod ten, bardzo krótki zresztą, wykorzystuje rozkaz maszynowy EOR identyczny z forthowym XOR.

Ciekawą właściwością XOR jest to, że wykonane na dowolnej potędze dwóch i liczbie od niej mniejszej daje jako wynik sumę tych liczb, bowiem bity jedynkowe nigdzie się tu nie spotykają. Być może, dało by się wymyślić praktyczne zastosowanie dla taj właściwości.

10.3 Bity ułatwiają wyszukiwanie

Omówione cechy operatorów logicznych sprawiają, że znalazły one zastosowanie w pamięciooszczędnych algorytmach wyszukiwania danych. Można wykorzystywać nie całe bajty, lecz pojedyncze bity jako nośniki informacji pomocnej w rozpoznawaniu elementów badanego zbioru. Metoda ta nosi nazwę mapy bitowej. Każdy bit może wskazywać, czy element ma określoną cechę. czy jej nie ma, czy spełnia, czy też nie ustalony warunek itd.

Wyobraźmy sobie instytucję zajmującą się zamianą mieszkań, która pragnie spośród posiadanego zbioru informacji o oferowanych mieszkaniach wyodrębniać te, których cechy mogłyby zadowolić klienta pragnącego zamienić mieszkanie. W jednym bajcie

można w takim wypadku zgromadzić informacje o ośmiu cechach posiadanych lub nie przez mieszkania. Na przykład możliwy jest wybór następujących cech:

Bit	nr	=1, jeżeli	=0, jeżeli
0	centralne ogrzewanie	jest	nie ma
1	gaz	jest	nie ma
2	centrum miasta	tak	nie
3	mieszkanie w bloku	tak	w domu wolnostojącym
4	garaż	jest	nie ma
5	blisko do sklepów	tak	nie
6	czy jest telefon	tak	nie
7	mieszkanie do drugiego piętra	tak	nie

Jest to, oczywiście, jedynie przykładowe przedstawienie cech. Można gromadzić je w większej liczbie bitów, przy czym np. trzy bity przeznaczyć na zapisywanie liczby izb.

10.4 Przykład: rozpoznajemy rasy psów

Metodę tę rozpatrzymy bliżej na przykładzie budowy programu pozwalającego na podstawie zaobserwowanych cech zewnętrznych określić rasę psa, którego zobaczyliśmy. Oto jedna z możliwości zakodowania ośmiu dostrzeżonych cech na podstawie odpowiedzi "tak" lub "nie" na kolejne pytania. Łatwo dostrzec, że niejednokrotnie - i to nie tylko w omawianym tu przykładzie - trudności sprawiać może jednoznaczne ustalenie obecności lub nieobecności jakiejś cechy. Przypiszmy kolejnym bitom właściwości ustalane z pomocą następujących pytań:

Bit	
0	Czy jest wyższy niż 60 cm?
1	Czy ma wydłużoną głowę?
2	Czy ma kwadratowy lub spłaszczony pysk?
3	Czy ma obfitą sierść?
4	Czy ma długi, miękki lub falisty włos?
5	Czy ma stojące lub przycięte uszy?
6	Czy ma długie uszy?
7	Czy ma mocną lub krępą budowę?

Zbudujmy tablicę, w której poszczególnym rasom przypiszemy kolejne cechy lub ich brak.

	>60	Głowa	Pysk	Sier.	Włos	Uszy	st.	dł.	Bud.	Dec
Bokser	1	0	1	0	0	1	0	1		165
Cocker	0	1	0	1	1	0	1	0		90
Dog	1	1	0	0	0	1	0	1		163
Doberman	1	1	0	0	0	1	0	0		35
Bernardyn	1	0	1	1	1	0	1	1		221
Owcz.alz.	1	1	0	1	0	1	0	1		171
Foksterier	0	1	1	1	0	1	0	0		46
Seter	1	1	0	1	1	0	0	0		27
Jamnik	0	1	0	0	0	0	1	1		194
Ratler	0	0	0	0	0	1	0	0		32
Chow-chow	0	0	1	1	0	1	0	1		172
Pekińczyk	0	0	1	1	1	0	0	0		28

W wyniku dokonanej operacji przypisania uzyskaliśmy dla wybranych 11 psów wartości bajtów przechowujących ich cechy.

Na tej podstawie można przystąpić do budowy programu. Pierwszą czynnością jest umieszczenie na jednym z ekranów dyskietki listy nazw psów. Można ją uzupełnić cechami psów oraz wartościami bajtów przechowujących cechy. Lista służyć będzie do tego, by w chwili ustalania nazwy psa program mógł z pomocą .LINE odczytać właściwą linię, zawierającą nazwę i ewentualne bliższe dane.

Wybierzmy na ten cel ekran nr 25 jednej z dyskietek. Najwygodniej jest umieścić listę jako komentarz ujęty w nawiasy okrągłe. Jednakże w wielu systemach nawias otwierający linię działa tylko do jej zakończenia. Niezbędne jest wówczas otwarcie każdej linii nawiasem. Pozostawiamy pomysłowości Czytelników środki pozbycia się go w chwili druku na ekranie czy innym urządzeniu.

Po wpisaniu nazw ekran 25 będzie wyglądał następująco, z pominięciem nawiasów w każdej linii:

```
SCR  #  25
 0       ( PSY RASOWE
 1      bokser              165
 2      cocker-spaniel       90
 3      dog                 163
 4      doberman             35
 5      bernardyn           221
 6      owczarek alzacki    171
 7      foksterier           46
 8      seter                27
 9      jamnik              194
10      ratler               32
11      chow-chow           172
12      pekińczyk            28
13      -- >
14
15
```

Do wygodnego posługiwania się programem powinniśmy stworzyć na następnym ekranie listę pytań wraz z ich numerami, które posłużą potem do wstawiania bitów w odpowiednich miejscach bajtu sprawdzającego zbieżność cech. Oto możliwy wygląd takiego ekranu:

```
SCR  #  26
0       ( CECHY PSÓW
1        C z y  m a:
2      1 więcej niż 60 cm wzrostu?
3      2 wydłużona głowa
4      3 kwadratowy lub plaski pysk?
5      4 obfita, sierść?
6      5 długi włos?
7      6 stojące lub przycięte uszy?
8      7 długie uszy?
9      8 krępa lub mocna budowę?
10     -->
11
12
13
14
15
```

Ogólne zasady budowy i działania programu są następujące: z pomocą nowego słowa CARRAY zdefiniowana została tablica liczb 1-bajtowych CECHY. Zgromadziliśmy w niej liczby, w których zakodowane są cechy poszczególnych psów, utrzymując kolejność taką, jak na ekranie. Zmienna BAJT służyć będzie do ustawiania w niej na 1 bitów dla cech numerowanych od 1 do 8 w przypadku, gdy odpowiedź na zadane pytanie jest twierdząca.

Kolejną czynnością będzie stworzenie mechanizmu, który na podstawie danych wprowadzanych przez nas z klawiatury w odpowiedzi na pytania z ekranu 26 wypełni bajt BAJT właściwą kombinacją bitów. Głównym instrumentem tej czynności jest słowo OR.

Ostatnia wreszcie czynność - to porównanie wyniku odpowiedzi z danymi o psach zakodowanymi w tablicy CECHY. Do wykonania tej pracy posłuży drugi ważny operator logiczny, XOR. Operator ten ustali, które z psów spełniają kryteria podane przez nas z klawiatury.

Pierwszą część zadania wykonują słowa BITY i WLASNOSCI. Jak działają BITY? Gdy podamy z klawiatury, że , powiedzmy, w przypadku bernardyna odpowiedź na pytanie nr 6 brzmi "tak", BITY tyle właśnie razy pomnożą jedynkę przez dwa i dla skorygowania wyniku działania pętli DO...LOOP podzielą go przez dwa. Powstająca w ten sposób liczba jest jedną z potęg dwóch, ma więc tylko jeden bit ustawiony na 1. Zostanie on z pomocą OR "dołączony" do bajtu BAJT.

Słowo WLASNOSCI zasila BITY kolejnymi danymi z klawiatury. Jest ono gotowe przyjąć wszystkie nasze kolejne informacje o pozytywnych odpowiedziach na pytania. Odbiór danych z klawiatury zapewnia sekwencja słów QUERY BL WORD HERE NUMBER DROP, którą poznaliśmy w swoim czasie w słowie INPUT. WLASNOSCI przyjmują od nas dane o numerach kolejnych odpowiedzi twierdzących w ten sposób, że po każdej liczbie będącej numerem pytania na ekranie naciskamy Return, a program czeka na podanie kolejnego numeru. By spowodować zakończenie działania pętli BEGIN... WHILE...REPEAT, w słowie tym przewidziano kontrolę, czy podawane liczby są dodatnie. Aby zakończyć wprowadzanie danych, należy napisać zero i Return.

Drugą część zadania wykonuje słowo DAJ. Czyni to porównując całość przekazanych przez nas danych, które zgromadzone zostały w zmiennej BAJT, z kolejnymi liczbami w tablicy CECHY. Ponieważ w naszym przykładowym programie umieściliśmy dane o 12 psach, tyle właśnie razy z pomocą pętli DO...LOOP słowo DAJ wykona operacje na zmiennej BAJT i kolejnych liczbach z tablicy CECHY.

Wszystkie omówione tu procedury zgromadzone zostały w słowie PSY, którego nazwa służy jako wywołanie programu. Ma zakończenie następuje wydrukowanie wszystkich nazw psów, odpowiadających podanemu przez nas wzorcowi bitowemu.

Pełny listing programu należy wgrać na kolejny ekran, w naszym przypadku 27, i załadować wszystkie trzy ekrany do pamięci. Pomocnicze słowo X służy do umieszczania danych w tablicy CECHY. Słowo HOME należy zdefiniować odpowiednio do zasad kasowania ekranu w danym komputerze. W Atari czyni to, przypomnijmy, 125 EMIT.

Na końcu rozdziału zamieszczony jest listing programu. Poprzedźmy go kilku ogólniejszymi uwagami. Program przedstawia szkielet metody, którą można w FORTH stosować do rozmaitych celów. W pełni możliwa jest wydatna rozbudowa przedstawionego tu klucza przez zwiększenie zarówno liczby ras, jak przede wszystkim liczby kontrolowanych cech, które mogą być rozmieszczone w większej liczbie bitów i bajtów.

Praktycznie nieograniczone jest pole stosowania podobnych mechanizmów gromadzenia danych, oznaczania ich przy pomocy kodów bitowych i wyszukiwania z zastosowaniem operatorów logicznych FORTH-a.

Można zbudować na tej podstawie katalog książek, komputerową wersję znanego od pokoleń klucza Jana Rostafińskiego do oznaczania roślin w Polsce dziko rosnących, program do diagnozowania chorób na podstawie ich symptomów, kalendarz prac działkowych uporządkowany według czasu zabiegów oraz poszczególnych kwiatów, owoców, warzyw, krzewów itd.

Operator XOR w przedstawionym programie można zastąpić przez AND. Zróbmy to na próbę i zobaczmy, jak zmieni się działanie programu.

A oto listing:

```
: CARRAY <BUILDS ALLOT DOES> +  ;
        15 CARRAY  CECHY
: X  CECHY  C!  ;
165 0 X 98 1 X  163 2 X 35 3 X 221 4 X  171 5 X
46 6 X  27 7 X  194 8 X 32 9 X 172 10 X 28 11 X
        0 VARIABLE BAJT
: BITY  1 SWAP 0  DO 2 * LOOP 2 /  BAJT @ OR BAJT ! ;
WLASNOSCI 0 BAJT  !
        BEGIN QUERY  BL WORD HERE NUMBER DROP DUP
                MINUS 0<
        WHILE  1 SWAP BITY CR DROP
        REPEAT   ;
: DAJ BAJT @  12 0
        DO DUP I CECHY C@ XOR 0=
                IF I 1+ 25 .LINE CR THEH
        LOOP DROP  ;
: PSY HOME  12  1
        DO I  26  .LINE CR LOOP
        CR ." Napisz liczby:" CR WLASNOSCI
        CR ." To moze byc:" CR DAJ DROP ;
```

ROZDZIAŁ 11
NARZĘDZIA

11.1 Jak zmagazynować program?

FORTH rozporządza prostym i efektywnym mechanizmem współpracy z nośnikami zewnętrznymi, w tym przede wszystkim ze stacją dyskietek. Istnieje szereg słów zapewniających wygodny kontakt miedzy słownikiem i pamięcią komputera a dyskietką.

Wszystkie czynności związane z wczytywaniem danych do komputera i ich zapisywaniem na dyskietkę, a także taśmę, FORTH wykonuje z pomocą własnego zasobu słów, w wielu wypadkach bez pośrednictwa dyskowych systemów operacyjnych. Już pierwsze wersje języka opracowane przez Moore'a zakładały automatyczne wczytywanie interpretatora i słownika do pamięci, a także samoczynne mechanizmy korygowania zapisów na dyskietce. Cechy te zostały zachowane, a często wzbogacone we współczesnych systemach FORTH-a.

Najbardziej rozbudowane są w FORTH formy współpracy ze stacją dyskietek. Współpraca z urządzeniami posługującymi się taśmą jest utrudniona, poza pewnymi specjalistycznymi zastosowaniami, ponieważ odpowiednich zapisów należało by szukać ręcznie. Korzystając z magnetofonu można bez wątpienia dość dobrze poznać FORTH, trudniej jest jednak w nim pracować. Posiadanie stacji dyskietek uwalnia od kłopotów związanych z zapisywaniem programów, odczytywaniem ich, a także ewentualnym wywoływaniem z dyskietki potrzebnych danych w trakcie wykonywania programu.

Podstawową formą współpracy z nośnikiem zewnętrznym jest traktowanie go jako miejsca przechowywania tekstów źródłowych
czyli zapisów definicji słów w postaci nieskompilowanej. Można to uznać za pewien odpowiednik listingu w BASIC-u, chociaż formy korzystania z tekstu źródłowego są inne.

Jest on wytwarzany w jednym z buforów, (dwóch lub trzech w poszczególnych systemach) , do czego potrzebny jest chociażby minimalny edytor. Tekst źródłowy może być następnie przeniesiony na dyskietkę, przy czym podstawową jednostką zapisu jest ekran. Przypomnijmy, że we współczesnych wersjach ma on na ogół 1024 bajty i zajmuje osiem sektorów dyskietki.

Tekst źródłowy może być ściągnięty z dyskietki w dwóch celach: albo po to, by go dalej redagować, albo też po to, by spowodować jego skompilowanie i umieszczenie w postaci gotowej do wykonania na aktualnym szczycie słownika.

Są również słowa umożliwiające wczytanie do pamięci komputera pojedynczego sektora, a także pojedynczej 64-bajtowej linii spośród 16 numerowanych od 0 do 15, na jakie w czasie redagowania dzielony jest ekran. Pewnym utrudnieniem jest niejednolitość znaczenia niektórych pojęć. W fig-FORTH blok ma 128 bajtów, a ekran obejmuje 8 bloków. W standardzie FORTH-79, a także, jak zdaje się wynikać z dokumentacji, FORTH-83, blok równy jest wielkością ekranowi, czyli ma 1024 bajty. Ekrany i bloki są numerowane od zera.

Omówię teraz operatory podstawowego predefiniowanego słownika FORTH-a. Zasady pracy edytora przedstawione będą w następnym punkcie rozdziału.

Przypuśćmy, że chcemy napisać program i utrwalić go na dyskietce. Powinniśmy najpierw ustalić, który ekran na dyskietce możemy wykorzystać. Wygodnym narzędziem jest dostępne zazwyczaj słowo INDEX. Wyświetla ono zerowe linie kolejnych ekranów w obszarze wyznaczonych dolnym i górnym numerem granicznym: n1 n2 INDEX. Zwyczajowo w linii zerowej umieszczany jest tytuł określający treść ekranu. Jest on ujęty w nawiasy okrągłe, przy czym po pierwszym musi następować spacja. Nawiasy te zaznaczają fragmenty tekstu mające charakter komentarza i są pomijane przez interpretator podczas kompilacji. Nawiasami można posługiwać się do wpisywania komentarzy w dowolnych miejscach linii jest to zatem odpowiednik REM w BASIC-u.

Gdy wybierzemy już ekran, do jego ściągnięcia do buforu służy słowo LIST. Trzeba je poprzedzić numerem ekranu, np.

```
30 LIST
```

W tym momencie rusza stacja dyskietek i ekran jest wgrywany. Pojawia się on w postaci prostokąta podzielonego na 16 numerowanych linii z numerem ekranu u góry. Numer ten przechowuje zmienna SCR. Warto podkreślić, że numery ekranu i linii dopisywane są dla ułatwienia obsługi ekranu w chwili jego wczytywania, nie znajdują się natomiast na dyskietce i nie są na nią zapisywane, jak to się dzieje w przypadku numerów linii w edytorze BASIC-u. Umożliwia to wykorzystywanie edytora FORTH-a do zapisywania rozmaitych tekstów na dyskietce, a także np. do pisania programów w języku C, który nie ma własnego edytora, lecz wymaga takiego, w którym nie stosuje się numeracji linii.

Ekran może być wypełniony znakami odpowiadającymi w kodzie ASCII wartości 0 . Np. w Atari jest to znak kierowego serca. Wygodnie jest w takim wypadku wypełnić ekran spacjami, tym bardziej, że znak zerowy jest często zdefiniowany w FORTH jako słowo powodujące zatrzymanie kompilacji.

Możemy wydrukować również pojedyncze linie z dowolnego ekranu. Służy do tego słowo .LINĘ używane następująco:

```
numer-ekranu numer-linii .LINE
```

.LINE drukuje linię, której wymiary obliczy (LINE) , przy czym opuszczane są końcowe spacje.

Nieco odrębne zastosowanie mają słowa BLOCK i BUFFER. Podają one na stos adresy początkowe odpowiednio bloku o podanym numerze i buforu, w którym znajduje się blok. Jeżeli blok o danym numerze nie znajduje się W pamięci, BLOCK ściąga go, BUFFER natomiast nie. Z pomocą BLOCK możemy uzyskać na ekranie tekst z dowolnego sektora na dyskietce.

Mechanizm ten kontrolowany jest przez dwie zmienne. USE zawiera adres buforu poprzednio używanego, a PREV - dostępnego aktualnie. Przypomnijmy też o kilku stałych określających wymiary. C/L podaje długość linii wynoszącą zwykle 64 bajty, B/BUF - długość buforu wynoszącą w fig-FORTH 128 bajtów, B/SCR - liczbę bloków w ekranie wynoszącą w fig-FORTH 8. Wszystkie te wielkości są niezbyt istotne dla użytkownika, natomiast system wykorzystuje je do obliczeń.

Powróćmy do fazy, w której użyliśmy komendy LIST. Wgraliśmy ekran i zaczęliśmy w nim pisać program z pomocą edytora. Definicje napisane w edytorze nie są od razu kompilowane i umieszczane na szczycie słownika. Powiedzmy, że zapisaliśmy już cały ekran tekstami definicji i chcemy je przenieść na dyskietkę. Pisząc słowo UPDATE sygnalizujemy, że ekran ma obecnie inną treść niż w chwili, gdy był wgrywany do pamięci. Powoduję to automatyczne wgranie ekranu na dyskietkę, gdy bufor jest potrzebny do umieszczenia innego ekranu. Komendy edytora z reguły wywołują UPDATE.

Teraz z pomocą słowa FLUSH powodujemy wgranie ekranu z powrotem na dyskietkę. FLUSH w niektórych systemach ma nazwę SAVE-BUFFERS.

Tak przedstawia się podstawowy mechanizm współdziałania FORTH-a ze stacją dyskietek w czasie tworzenia źródłowego tekstu programu. Trzeba teraz przygotować jego wykonanie. Słowo LOAD poprzedzone numerem ekranu powoduje, że tekst źródłowy ekranu dyskietki jest kompilowany i umieszczany w słowniku. Dopiero w tej fazie interpretator rozpoczyna kontrolę poprawności tekstu, który wcześniej napisaliśmy, taką samą, jaką przeprowadza wtedy, gdy piszemy definicje wprost z klawiatury. Gdy kontrola ujawnia błąd we wgrywanym tekście, kompilacja zostaje przerwana. Należy wówczas przystąpić do poprawek z pomocą edytora. FLUSH poprawki te wprowadza na ekran dyskietki. Dobrze jest przyswoić sobie nawyk używania FLUSH w każdym przypadku poprawiania ekranu.

Z kolei używając VLIST warto sprawdzić, czy którekolwiek definicje zostały już skompilowane i są w słowniku. Jeżeli tak, to najpraktyczniej będzie skasować pierwszą definicję z ekranu, na którym wystąpił błąd, kasując tym samym wszystkie dalsze, po czym używając LOAD skompilować ponownie poprawiony ekran.

FORTH ma dwa dalsze użyteczne słowa. Mianowicie --> umieszczone za tekstem powoduje, że działalność LOAD przenosi się na następny ekran. Tak więc wystarczy poprzedzić LOAD numerem pierwszego ekranu zawierającego program, a wszystkie następne, jeżeli łączone są użyciem --> , będą

kolejno wgrane i zinterpretowane. Na końcu ekranu, na którym chcemy zakończyć kompilację, celowe jest umieszczenie słowa :S.

Słowo to może być również używane w programach i powoduje, na przykład, w konstrukcjach warunkowych, wcześniejsze przerwanie wykonania słowa, ma zatem podobne działanie, jak LEAVE w pętlach liczonych, tyle że wyprowadza nas poza końcowy średnik słowa, a nie tylko za pętlę. W niektórych implementacjach istnieje słowo EXIT wykonujące w omawianych sytuacjach analogiczną funkcję.

Jeszcze jedno ważne słowo - to EMPTY-BUFFERS (opróżnij bufory). Możemy z jego pomocą oczyścić bufory, po czym ponownie wylistować ekran, na przykład, w celu jego przeredagowania. Pamiętać należy o tym, by poprzedzić EMPTY-BUFFERS poleceniem FLUSH, jeżeli chcemy utrwalić wcześniejsze poprawki.

W buforach można równocześnie mieć, zależnie od systemu, dwa lub trzy ekrany. Wtedy można je wywołać na zmianę, redagować, a także wykorzystując możliwości edytora, ewentualnie numerację ekranów, przenosić teksty z jednego ekranu na inny itd. niektóre systemy mają słowo TRIAD, pozwalające wyświetlić trzy kolejne ekrany.

Odrębny problem - to korzystanie ze słowa R/W:

```
adr blk f R/W
```

Jego działanie zależy, od wartości f. Gdy f=0, następuje zapisywanie do bloku dyskietki o numerze blk danych spod adresu adr. Gdy f=1, następuje wczytywanie zawartości sektora blk do pamięci od adresu adr. Stosując R/W w pętli można wczytywać i zapisywać dowolną liczbę sektorów. Można również użyć FORTH-a do kopiowania programów i dyskietek o dowolnym charakterze.

11.2 Edytor

Edytor nie jest niezbędną częścią podstawowego systemu FORTH-a, dość trudno jest jednak obejść się bez niego w poważniejszych zastosowaniach. Dlatego systemy FORTH-a zawierają z reguły edytor. Istnieją rozmaite typy edytorów - od naj-

prostszych, umożliwiających podstawowe operacje do bardzo rozbudowanych.

Edytor może. być w całości zdefiniowany z pomocą podstawowego zasobu słów FORTH-a bez konieczności uciekania się do definicji w kodzie maszynowym. Dlatego możliwe jest zbudowanie edytora, który będzie działał na wszystkich komputerach.

W aneksie A.3 Czytelnik znajdzie komplet definicji takiego edytora wraz ze szczegółowym opisem działania słów. Jest to edytor stosunkowo prosty, zawiera jednak wiele użytecznych operacji, takich jak dokładne wskazanie miejsca błędu, który spowodował wstrzymanie kompilacji, wypełnienie ekranu spacjami, kopiowanie jednego ekranu na inny itd.

Na tym miejscu przedstawię jedynie ogólne zasady działania edytora. Posługuje się on wszystkimi operatorami słownika podstawowego omówionymi w poprzednim punkcie, a jednocześnie umożliwia wpisywanie tekstów na ekran. Teksty przechowywane są czasowo w buforze pomocniczym znajdującym się pod poznanym już przez nas adresem PAD, nad szczytem słownika.

Mogą być stamtąd przeniesione albo wprost do wskazanego z pomocą słowa P numeru linii, albo wprowadzone słowem edytora I do którejkolwiek linii ekranu. Wyświetlenie od nowa tekstu ekranu, nad którym pracujemy, ułatwia słowo L nie wymagające numeru ekranu. Jeżeli chcemy przejść na inny ekran o numerze n, znajdujący się w pamięci, trzeba użyć n LIST albo wprowadzić numer tego ekranu do zmiennej SCR.

Korzystanie z edytora wymaga opanowania pewnych nawyków i nabycia wprawy. Najlepiej jest przeznaczyć , do ćwiczeń jakiś nieużywany ekran, oczyścić go, wpisać z pomocą P kilka linii i wypróbowywać działanie rozmaitych komend.

Podczas pracy w edytorze możemy w każdej chwili przejść na tryb bezpośredniego wykonywania komend, na przykład po to, by przeprowadzić jakieś obliczenia lub zbudować pomocniczą definicję i sprawdzić jej działanie. Opuszczenie edytora może się zdarzyć również na skutek naszej omyłki, jeżeli, powiedzmy, zapomnimy wpisać P po numerze linii, którą redagujemy FORTH odczyta to jako tekst do bezpośredniej interpretacji, co może powodować błędy.

Należy również pamiętać, że po chwilowym wyjściu z edytora powrót do niego wymaga nieraz ponownego wywołania podsłownika przez napisanie EDITOR.

11.3 Asembler

W porównaniu z edytorem podsłownik asemblera jest bardziej autonomiczną częścią systemu. Opiera się w głównej mierze na własnym zasobie słów, których definicje są specyficzne dla poszczególnych mikroprocesorów i komputerów.

Praktycznie wszystkie "szanujące się" systemy FORTH-a zawierają asembler, ponieważ także dla średnio zaawansowanego programisty jest on narzędziem nad wyraz użytecznym, pozwala na szybkie i proste wprowadzenie do FORTH-a słów definiowanych w kodzie maszynowym.

Punktem wyjścia do budowy asemblera jest zdefiniowanie słowa CODE, którego można dokonać następująco:

```
: CODS  [COMPILE]  ASSEMBLER CREATE SMUDGE ;
```

Zasada dalszego budowania asemblera jest prosta: wszystkie mnemoniczne nazwy rozkazów danego asemblera definiuje się jako słowa FORTH-a o takich samych lub podobnych nazwach, na przykład, z dopisaniem na końcu przecinka. W asemblerze FORTH-a można wykorzystać efektywnie szereg konstrukcji tego języka, jak struktury warunkowe i pętle. W efekcie asembler FORTH-a może zawierać wiele możliwości, jakimi rozporządzają rozbudowane makroasemblery, a jednocześnie jest od nich znacznie prostszy w użyciu dla tego, kto poznał już programowanie w FORTH.

Asembler wywoływany jest przez napisanie nazwy podsłownika ASSEMBLER, a w wielu systemach także automatycznie słowem CODE. Praca w nim odbywa się przy użyciu odwrotnej notacji polskiej. Tak więc zamiast, na przykład

LDA 155 pisze się 155 LDA,

Inny jest także sposób posługiwania się trybami adresowania. Ogólna postać słów definiowanych w asemblerze jest następująca:

: CODE nazwa operand tryb instrukcja [operand tryb instrukcja] itd.

Złożoność asemblera FORTH-a zależy, rzecz prosta, od złożoności kodu maszynowego w danym komputerze, liczby rozkazów, liczby trybów adresowania itd. Wobec ogromnego zróżnicowania tych cech nie jest możliwe w tej książce ze względów objętościowych przedstawienie i skomentowanie konkretnego asemblera FORTH-a.

11.4 Grafika, dźwięk, programy użytkowe

Z tych samych względów, co przedstawione przed chwilą, możemy jedynie bardzo zwięźle omówić sposób posługiwania się grafiką i dźwiękiem w FORTH oraz sprawę wyspecjalizowanych pakietów użytkowych w tym języku.

Jest oczywiste, że FORTH rozporządza ogromnymi możliwościami w dziedzinie grafiki komputerowej. W przeciwnym wypadku nie odgrywałby takiej roli w projektowaniu gier, jaką spełnia. Podstawowe procedury graficzne, zwłaszcza gdy chodzi o grafikę o wysokiej rozdzielczości, powinny być programowane w asemblerze FORTH-a. W przeciwnym wypadku czas ich wykonania może być zbyt długi. Jednakże znając adresy podprogramów sterujących w danym komputerze zmianą trybów graficznych, kolorów i innymi elementami grafiki można także bez asemblera zbudować całkiem znośnie działające narzędzia pomocne we wzbogaceniu graficznej strony naszych programów w FORTH.

Dużym ułatwieniem jest to, że część komputerów, jak Atari i Commodore, ma podprogramy do generowania ruchu, kształtów obiektów ruchomych i innych elementów użytecznych w grach. Zachęcamy wszystkich miłośników eksperymentu do wykorzystania tych narzędzi w celu tworzenia ciekawych rozwiązań graficznych i animacji.

Prościej przedstawia się na ogół generowania dźwięku. W aneksie A4 Czytelnik znajdzie definicje do sterowania dźwiękiem na Atari. Wzorując się na nich pomysłowy programista może łatwo zbudować podobne narzędzia dla innego komputera.

ROZDZIAŁ 12
PROGRAMY

Kilka stosunkowo prostych programów zamieszczonych w tym rozdziale może być pomocnych w nabyciu praktycznej umiejętności samodzielnego programowania w FORTH. Źródłem inspiracji do większości programów były prace [3] i [5] , przy czym propozycje w nich zawarte zostały przetworzone i opatrzone własnym komentarzem.

12.1 Wieże z Hanoi

XIX-wieczny wybitny matematyk francuski Edouard Lucas zaczerpnął, być może, pomysł do tej łamigłówki z legendy hinduskiej zmieniając miejsce; Benares na Hanoi. Oto treść tej legendy według opisu de Parville'a.

W miejscu, gdzie znajduje się środek Ziemi, Brahma ustawił na brązowej tabliczce trzy diamentowe pałeczki o wysokości łokcia i grubości, tułowia osy. Na jedną s nich przy stworzeniu świata nanizał 64 krążki z czystego złota o średnicach malejących ku górze. Kapłani dniem, i nocą zajęci są przenoszeniem krążków z pierwszej pałeczki na trzecią posiłkując się drugą według następujących zasad: za jednym razem wolno przenieść tylko jeden krążek; nie można kłaść krążka większego na mniejszy. Gdy kapłani zakończą pracę, nadejdzie koniec świata.

Nie musimy się go obawiać. Szczepan Jeleński w "lilavati" wskazuje, że przenosząc co sekundę krążek kapłani poświęciliby na to pięć miliardów stuleci. Istotnie, liczba przeniesień dla n krążków wynosi 2^n-1 czyli nieomal podwaja się po dodaniu jednego krążka.

Wieże z Hanoi zrobiły karierę w matematyce i informatyce. Są mianowicie łamigłówką, którą stosunkowo łatwo można rozwią-

zać przy zastosowaniu rekurencji, natomiast znacznie trudniej metodą iteracji, pętli. Stały się zatem klasycznym przykładem algorytmu rekurencyjnego. Został on zrealizowany w wielu językach programowania.

Algorytm można opisać następująco. Jeżeli przez A, B i C oznaczymy pałeczki, to mając n krążków, np. 8, należy przenieść n-1 krążków z A na B zostawiając C wolnym. Teraz przypiszmy pałeczce B oznaczenie A, a pałeczce A B. Kolejne zadanie polega na tym, by znów z A na B przenieść n-2 krążki. W ten sposób przez zmianę oznaczeń pałeczek A i B doprowadzimy do tego, że na pałeczce C przybywać będzie za każdym powtórzeniem tego samego programu po jednym krążku aż do wykonania zadania.

Jego realizacja w FORTH wymaga zdefiniowania słowa, kopiującego trzy liczby na szczyt stosu. Było ono przedmiotem ćwiczenia 5.

```
: 3DUP >R 2DUP R ROT ROT R>  ;
```

By wykonać zadanie, przypiszemy pałeczkom liczby 1, 2 i 3. Zawsze zachowana będzie relacja A+B+C=6. Przenoszenie krążków z A na B i z B na C przygotowują procedury:

```
:  TAM                              ( a c n --- a c n a b n-1 )
        3DUP                        ( a c n a c n )
        ROT  ROT  OVER  +           ( a c n n a c+a )
        6 SWAP  -  ROT  1  -  ;     (a c n a b n-1 )
: Z.POWROTEM                        ( a c n --- a c n b c n-1 )
        3DUP                        (a c n a c n )
        SWAP  ROT  OVER  +          ( a c n n c c+a )
        6 SWAP  - SWAP  ROT  1 - ;  (a c n b c n-1)
```

Konieczne jest zdefiniowanie słowa drukującego informacje, których pałeczek dotyczy kolejne przeniesienie krążków:

```
:  DRUKUJ
        3DUP   DROP   SWAP  .  ." NA "  .  ."   ,"   ;
```

Jedna informacja ma długość ośmiu znaków, toteż gdy liczba znaków w wierszu ekranowym podzielna jest przez 8, wyniki drukowane będą w równym szeregu bez przeniesień. Można też na końcu DRUKUJ dopisać CR, by każde przeniesienie opisane zostało w nowym wierszu.

Główna procedura rekurencyjna ma poniżej przedstawioną postać. Użyto w niej metody omówionej w punkcie 8.5.

```
:   WIEZE              ( a c n ---)
        [ SMUDGE ]
-DUP     IF            ( a - n  )( sprawdzenie czy n<>0)
        TAM            ( a c n a b n-1  )
        WIEZE          ( a c n  )
DRUKUJ                 ( --- )
Z.POWROTEM             ( a c n b c n-1 )
        WIEZE          ( a c n )
            THEN DROP DROP DROP
[ SMUDGE ] ;
```

WIEZE wymagają wprowadzenia na stos liczbowych odpowiedników pałeczek A (1) i C (3) oraz liczby pałeczek n. By nie podawać za każdym razem dwóch pierwszych liczb, stwórzmy , na koniec

```
: HANOI 1 3 ROT CR WIEZE ;
```

Wystarczy napisać, na przykład, 6 HANOI, by dla sześciu krążków otrzymać opis wszystkich kolejnych przeniesień. Gdy je wykonamy z krążkami lub kartkami ułożonymi w malejącym porządku, przekonamy się, jak pomocny w rozwikłaniu zagadki stworzonej przez Lucasa sto lat temu może dziś być komputer. Nie próbujmy mu jednak zlecić zadania, które postawił mnichom Brahma.

12.2 Osiem hetmanów

Jak ustawić na szachownicy osiem hetmanów tak, by żaden z nich nie mógł pobić innego? Także tę zagadkę można rozwikłać

z pomocą algorytmu rekurencyjnego, jest on jednak bardziej skomplikowany niż dla wież z Hanoi.

Hetman bije w ośmiu kierunkach na dowolną odległość aż po krawędź szachownicy. Każdego hetmana trzeba zatem ustawić w innej linii poziomej i kolumnie szachownicy. Nie mogą one ponadto stać na tych samych przekątnych. W ogólnej postaci zadanie polega na tym, by ze zbioru wszystkich możliwych rozstawień wybrać rozstawienia spełniające nasz warunek.

Zadanie po raz pierwszy sformułował w r. 1848 szachista niemiecki M. Betzel, a jego rodak, niewidomy od urodzenia profesor Franz Nauk znalazł wszystkie rozwiązania i ogłosił je w r. 1850. Dziś zadanie ma bardzo obszerną literaturę, zajmowało się nim wielu wybitnych matematyków, wśród nich Carl Gauss. W informatyce należy dziś do zadań klasycznych.

Jego algorytm można opisać następująco. Aby ustawić N hetmanów na szachownicy, na której znajduje się już 8-N hetmanów należy: znaleźć pierwszy rząd dostępny dla hetmana, znaleźć pierwszą kolumnę dostępną dla hetmana, ustawić hetmana na polu, rozmieścić N-1 hetmanów na szachownicy. Czynimy to, dopóki nie będzie już więcej hetmanów do ustawienia.

Ustanawiamy zmienną ROZWIAZANIE, która będzie przechowywać numer rozwiązania. Tworzymy słowo definiujące WEKTOR, które podawać będzie adres i-tej komórki, po czym z jego pomocą jednowymiarową tablicę WSPOLRZEDNA do przechowywania danych potrzebnych przy realizacji zadania.

```
0 VARIABLE ROZWIAZANIE

: WEKTOR <BUILDS HERE OVER ERASE ALLOT DOES>  +  ;

1024 WEKTOR WSPOLRZEDNA
```

WSPOLRZEDNA zawierać będzie dla kolejnych rozstawień liczbę hetmanów +1 oraz iloczyn numeru kolumny i-tego hetmana przez liczbę hetmanów, przy czym hetmany jeszcze nie ustawione będą miały kolumnę 0. Reszta danych będzie służyć do weryfikacji, czy kolumna jest większa od 0, a mniejsza od 9.

```
: LHET                          ( --- liczba hetmanów+1 )
        0 WSPOLRZEDNA C@ ;
: POSTAW                        ( rząd kolumna --- )
        SWAP WSPOLRZEDNA C! ;   ( umieszcza hetmana )
: USUN                          ( rząd --- )
0 POSTAW ;                      ( kasuje hetmana )
```

Kolejny krok - to zdefiniowanie słona WOLNE, które będzie sprawdzać, czy hetman postawiony w rzędzie "rząd" nie jest pod biciem hetmana "n" stojącego w rzędzie "n". Weryfikacja oparta jest na tym, że dla hetmanów stojących na przekątnej wznoszącej się suma numerów rzędu 1 kolumny jest stała, natomiast dla przekątnej schodzącej w dół stała jest różnica tych wielkości. Znacznik f będzie równy 1, gdy hetman nie jest pod biciem, a 0 w przeciwnym wypadku.

```
: WOLNE       ( rząd n --- rząd f )
2DUP = >R
OVER WSPOLRZEDNA C@ OVER WSPOLRZEDNA C@ -
ROT ROT - ABS OVER ABS = SWAP 0= OR 0=
        R> OR ;
```

Następne słowo dokonuje z pomocą WOLNE weryfikacji dla wszystkich dotychczas postawionych hetmanów. Jeżeli nowy hetman nie jest pod biciem żadnego z nich, wówczas f=i.

```
: WSZYSTKIE-WOLNE             ( rząd --- rząd f   )
1 OVER 1 DO OVER I WOLNE AND LOOP ;
```

Sprawdzenie, czy hetman nie "wypadł" poza szachownicę, dokonuje słowo BRZEG zostawiając f=1, gdy hetman jest w obrębie szachownicy:

```
: BRZEG                       ( rząd --- rząd f   )
LHET OVER WSPOLRZEDNA C@ > ;
```

Następne słowa wykonają czynności: drukowania kolejnych poprawnych rozstawień, inicjowania obliczeń oraz poszukiwania dalszych rozwiązań. Gdy znaczniki f przybierają wartość 1, kolejne sprawdzenia prowadzą do znalezienia właściwej pozycji hetmana.

```
: DRUK  ( --- )
CR  LHET  1  DO  I  WSPOLRZEDNA  C@  LHET  1
DO DUP  I =
IF  ." X "  ELSE ." . "  THEN
LOOP  DROP  CR
LOOP
1  ROZWIAZANIE +!  CR  ." Rozwiazanie nr "
ROZWIAZANIE ? CR ;
INIC  (--- )  ( inicjuje szachownice z hetmanami w rz. 0 )
LHET  1  DO I  USUN  LOOP  0  ROZWIAZANIE !  ;
NAPRZOD  ( rząd --- rząd f )  ( f=1, gdy hetmana w rzędzie
( "rząd" można przesunąć do następnej kolumny )
DUP  DUP  WSPOLRZEDNA  C@  1+  POSTAW  BRZEG
IF  1  ELSE DUP USUN 0  THEN ;
PIERWSZA-WOLNA  ( rząd --- rząd f ) ( f=1,gdy hetman może
( przejść do następnej kolumny w rzędzie )
BEGIN NAPRZOD
IF  WSZYSTKIE-WOLNE  -DUP  ELSE  0  1  THEN
UNTIL  ;
POZYCJA  ( rząd --- f )
[ SMUDGE ]
LHET  OVER >  IF
BEGIN  PIERWSZA-WOLNA  IF  1+ POZYCJA  -DUP  ELSE  1 - 0  1
THEN  UNTIL     ELSE DRUK  1 - 0 THEN
[ SMUDGE ] ;
HETMANY  9  0  SWAP POSTAW  INIC  1  POZYCJA 2 DROP  ;
```

By uruchomić program, należy napisać: HETMANY. Kilka chwil trwa inicjalizacja, po czym na ekranie pojawiają się kolejne rozwiązania w postaci szachownic, na których znaki "X" wskazują pozycje hetmanów.

12.3 Gra Life

A oto zadanie obmyślone całkiem niedawno, które szybko i chyba na trwałe zdobyło dużą popularność. Jest to znakomita gra symulacyjna, w której komórki rodzą się, trwają lub umierają. Nazwano ją oknem w inny Wszechświat.

Grę Life, czyli Życie, wynalazł w r. 1970 matematyk z Cambridge John Horton Conway, a upowszechnił artykułem w "Scientific American" znany popularyzator rozrywek matematycznych Martin Gardner.

Conway zaoferował 50 funtów nagrody temu, kto pierwszy zdoła potwierdzić lub obalić tezę, że populacja tworzona przez Life może rosnąć nieograniczenie. We wczesnych latach siedemdziesiątych w Massachusets Institute of Technology Bill Gosper zdołał osiągnąć to, że Life na komputerze PDP-1 utrzymywała swój zmienny Wszechświat rodzących się i umierających komórek przez 18 miesięcy.

Rozważania nad tą grą, jak pisze Gigi Bisson w miesięczniku "Antic" (luty 1986) , zapoczątkowały w USA rozwój badań w dziedzinie, które otrzymała potem nazwę Sztucznej Inteligencji. Świat "komputerowego darwinizmu", jaki tworzy Life, pociąga wielu miłośników informatyki. Być może, i my zdołamy znaleźć receptę na jego nieograniczone trwanie.

Zasady, którymi rządzi się Life, są następujące:

Każda komórka, która ma 2 lub 3 sąsiadów, żyje dalej.

Każda komórka mająca co najmniej 4 sąsiadów umiera z powodu przeludnienia, a każda, która ma mniej niż dwóch sąsiadów, umiera na skutek osamotnienia.

Każde wolne pole mające trzech sąsiadów daje życie nowej komórce. Narodziny i umieranie elementów populacji dokonuje się równocześnie na całym obszarze i za każdym razem powstaje nowe pokolenie.

Jak to zrealizować na komputerze? Należy, oczywiście, wyznaczyć ograniczono pole rozwoju populacji, naturalną formą jej przedstawienia będzie siatka o wymiarach dim X, dim Y, w której polach gwiazdką oznaczymy komórki zamieszkałe, a spacją puste, Należy zapobiec przenikaniu się pokoleń, toteż przed narodzinami kolejnego pokolenia siatka musi być oczyszczona. Pierwszą myślą jest zbudowanie dwóch siatek, ale wymagałoby to większego zużycia pamięci.

Elegantszym rozwiązaniem jest kodowanie w każdym polu siatki nie tylko obecności lub nieobecności elementu pokolenia N, lecz także danych o jego przyszłości w pokoleniu N+1.

Pokolenie N	Pokolenie N+1
Istnieje: 1	Śmierć: 5
	Przeżycie: 1 lub 3
Nie istnieje: 0	Narodziny: 2
	Bez zmian: 0 lub 4

Innymi słowy, komórka będzie zajęta w pokoleniu N+1, jeżeli jej wartość po skasowaniu pokolenia N wyniesie 1, 2 lub 3. W pozostałych przypadkach komórka będzie pusta. Jak widać, będziemy posługiwać się niedużymi wartościami, co pozwala na zastosowanie tablicy dla wartości jednobajtowych przedstawionej już w punkcie 9.1 . Powtórzmy ją:

```
:  ARRAY
        <BUILDS          ( dimX dimY --- )
              OVER  ,    ( umieszcza. dimY pod adresem pfa+2 )
              * ALLOT    ( rezerwuje dimX*dimY bajtów )
        DOES>
              DUP @      ( umieszcza dimX na stosie )
              ROT *      ( oblicza dimY*dimX )
              +  +  2+   ( oblicza, bezwzględny adres komórki )
```

Wskaźniki tablicy będą przybierały wartości od 0 do dimX-1 i dimY-1. Z kolei definiujemy z pomocą ARRAY tablicę UNIVERSUM:

```
dimX dimY ARRAY UNIVERSUM
```

Ramowy szkielet programu możemy wyrazić z pomocą definicji, którą wprowadzimy do programu dopiero po zdefiniowaniu składających się na nią procedur. Przed jej użyciem trzeba wprowadzić na stos liczbę pokoleń.

```
: GRA 0  DO
DRUK    (  wyświetla pokolenie N )
PRZYGOTOWANIE
( gasi pokolenie N i zostawia dane dla następnego )
TWORZENIE  ( tworzy pokolenie N+i )
LOOP
DRUK  ;    ( drukuje ostatnie pokolenie )
```

Do manipulowania tablicą UNIVERSUM służyć będą słowa:

```
: WSTAW    ( indX indY --- )
      UNIVERSUM  1  SWAP  C!  ;
: USUN     ( indX indY --- )
      UNIVERSUM 0 SWAP C!  ;
```

Wyzerowanie całej tablicy spowoduje słowo ZERUJ. Użyty jest w niej wskaźnik J, który przed użyciem ZERUJ należy zdefiniować tak, jak w punkcie 5.6.

Wybrane przez nas wymiary pola : poziomy - dimX i pionowy - dimY zdefiniujemy jako stałe o takich właśnie wymiarach:

```
dimX -CONSTANT DIMX   dimY CONSTANT DIMY
```

W niektórych komputerach może występować odwrotna kolejność.

Oznaczenia "dimX" i "dimY" musimy, oczywiście, zastąpić

liczbami nie przekraczającymi wymiarów ekranu, które nie są jednakowe w rozmaitych komputerach.

```
: ZERUJ
    DIMY  0 DO
       DIMX 0 DO
    I  J  USUN
       LOOP
LOOP  ;
```

Słowo DRUK zaczyna się od czynności oczyszczenia ekranu. W Atari wykonuje ją 125 EMIT, ale w innych komputerach wartość ta, jak o tym była mowa w punkcie 5.6 , jest inna. Należy ją ustalić i odpowiednio zdefiniować HOME.

```
: DRUK   ( --- )
HOME  DIMY  0  DO  DIMX  0  DO
    I  J    UNIVERSUM C@
IF  ."  *"  ELSE  SPACE  THEN
    LOOP  CR  LOOP  ;
```

Sercem algorytmu jest PRZYGOTOWANIE. Wykonuje ono następujące czynności:

- kasuje wszystkie elementy siatki;
- w stosunku do każdego elementu wyodrębnia kwadrat 3 na 3 otaczających go komórek, biorąc także pod uwagę sąsiedztwo brzegu pola gry;
- oblicza sumę wszystkich elementów w kwadracie otaczających pól w pokoleniu N mających wartość 1,3 lub 5, czyli nieparzystych:
- usuwa 1, jeżeli badany element istnieje w pokoleniu N;
- koduje odpowiednią wartość rozpatrywanego elementu.

Pomocnicze słowo WERYFIKACJA na podstawie liczby sąsiadów w pokoleniu N określa przyszłość komórki w pokoleniu N+1.

```
:   WERYFIKACJA  ( liczba stanu sąsiadów --- )
DUP 3 = ( liczba stanu f )
IF
DROP  2+
ELSE
2 =  0=  ( stan stan nr2 )
IF 4 + THEN
THEN  ;
: PRZYGOTOWANIE  ( --- )
DIMY 0 DO
 DIMX 0 DO
0  J  2+ DIMY MIN  J  1  - 0    MAX
DO J  2+ DIMX  MIN J  1  - 0 MAX
 DO I J UNIVERSUM C@  1  AND +  1+
 LOOP
LOOP
I  J UNIVERSUM C@  1  AND  ( liczba sąsiadów )
 SWAP OVER - WERYFIKACJA  ( koduje przyszłość )
I  J  UNIVERSUM C!  1+
 LOOP
LOOP  ;
```

Poniższa tabela określa skutki działania WERYFIKACJA:

POKOLENIE				POKOLENIE N+1	
N	W=0	W=1	W=2	W=3	W > 3
Stan=1 Zajęta	Stan=Stan+4=5 śmierć przez izolację	Bez zmian		Stan=Stan+2=3 przeżycie	Stan=Stan+4=5
Stan=0 Pusta	Stan=Stan+4=4 bez zmian	Bez zmian		Stan=Stan+2=2 narodziny	Stan=Stan+4=4 bez zmian

Ostatnia procedura kończy tworzenie nowego pokolenia.

```
: TWORZENIE  DIMY  0  DO  DIMX  0  DO
          I  J  UNIVERSUM  DUP  C@  DUP  3  >
  IF  DROP  0  ELSE  DUP  1  >  IF  DROP  1  THEH  THEN
SWAP  C!    ( wprowadza nową wartość )
                              LOOP
                LOOP ;
```

Po wprowadzeniu początkowego układu elementów wywołując n GRA otrzymamy obraz n pokoleń, jeżeli wcześniej nie umrą wszystkie elementy naszego komputerowego "Wszechświata".

Program tu przedstawiony pozwala zobaczyć Life w działaniu, obserwować pojawianie się i znikanie elementów populacji w każdym kolejnym pokoleniu.

Kilka rad i wskazówek dotyczących jego wykorzystania. Pożądane jest ograniczenie wielkości pola, ponieważ wzrasta wówczas szybkość zmiany pokoleń. Zdefiniowanie DIMX i DIMY na poziomie ok. 10 jest do pierwszych prób zupełnie wystarczające. Wartość taką można podać przy definiowaniu tych stałych. Zmiany wartości dokonuje się z pomocą:

```
liczba  ' stała    !
```

Kolejny krok - to skasowanie pola z pomocą SKASUJ. Następnie posługując się WSTAW rozmieszczamy elementy. Ich układ można sprawdzić wywołując go z pomocą DRUK. Następnie uruchamiamy grę pisząc:

```
n  GRA
```

gdzie n oznacza liczbę pokoleń.
Program w przedstawionej tu postaci stanowi trzon, który można obudować dodatkowymi słowami, na przykład, do wygodnego wprowadzania danych z pomocą słowa INPUT poznanego w punkcie 6.3. Gdy rozporządzamy pakietem grafiki, pole może być przedstawiane w postaci powiększonej.
Szczególnie celowa jest wprowadzanie mechanizmu umożliwiającego przerwanie wykonania programu w dowolnej chwili.

Można do tego wykorzystać słowo ?TERMINAL, na przykład:

```
: GRA1  BEGIN ?TERMINAL  IF  ;S THEN
DRUK PRZYGOTOWANIE TWORZENIE LOOP  DRUK ;
```

Gdy naciśniemy dowolny klawisz, np. spacje, i przytrzymamy go do pojawienia się kolejnego pokolenia, gra się zakończy. Można ją jednak ponownie wywołać, jeżeli chcemy zbadać dalsze losy układu pisząc np. 20 GRA1.

12.4 Liczby zespolone

Szerokie zastosowanie w rozmaitych dziedzinach nauki i techniki znajdują liczby zespolone. Mają one postać

a + b * i

gdzie a i b są dowolnymi dwiema liczbami rzeczywistymi, natomiast i jest wartością szczególną, mianowicie $i^2 = -1$. O liczbie zespolonej mówimy, że ma część rzeczywistą a i część urojoną b. Określenie "urojona" zrodziło się w dawnych wiekach, kiedy to z przesadną bojaźnią odnoszono się do pojęcia pierwiastka z liczby ujemnej, którego dokładnego znaczenia nie umiano wyjaśnić.

Geometrycznie liczbę zespoloną z=x+yi interpretuje się jako punkt na płaszczyźnie mający współrzędne prostokątne x,y.

Na liczbach zespolonych można wykonywać działania arytmetyczne. Wymaga to stworzenia odrębnej struktury danych odznaczającej się następującymi cechami: każda liczba zespolona przedstawiana jest jako para liczb rzeczywistych; dostęp do zmiennej zespolonej odbywa się za pośrednictwem jej nazwy, której wywołanie powoduje umieszczenie na stosie adresu jej części urojonej z częścią rzeczywistą następującą po niej w pamięci.

Wymaga to utworzenia słowa definiującego zmienne zespolone:

```
    : COMPLEX
        <BUILDS
        ,   ,   ( wprowadza do słownika 2 wartości ze stosu )
        DOES> ; ( potrzebny adres jest już na stosie )
```

Rz - oznacza wartość części rzeczywistej, a ur - urojonej.

Możemy teraz wykorzystać to narzędzie w celu uzyskania dostępu do zmiennej zespolonej z pomocą słów X@ i X!.

```
: X@        ( adr --- rz    ur )
    DUP     ( adr dr )
    @       ( adr ur )
    SWAP    ( ur adr )
    2+      ( ur adr+2 )
    @       ( ur rz )
    SWAP    ( rz ur )
```

A oto cztery słowa służące do działań arytmetycznych na liczbach zespolonych. Wykonują one kolejno: dodawanie, odejmowanie, mnożenie przez liczbę rzeczywistą i mnożenie dwóch liczb zespolonych.

```
: X!    ( rz ur adr --- )
    SWAP OVER  !  2+  !  ;
: X+      (  rz1 ur1 rz2 ur2 --- rz1+rz2 ur1+ur2 )
    ROT  ( rz1 rz2 ur2 ur1 )
    +    ( rz1 rz2 ur1+ur2 )
    >R   ( rz1 rz2 )
    +  R>  ;  ( rz1+rz2 ur1+ur2 )
: X-  ROT  SWAP  -  >R  -  R>  ;
: X*'    ( rz ur rzecz --- rz*rzecz ur*rzecz )
    SWAP  OVER  ( rz rzecz ur rzecz )
    *  ROT  ROT * ( ur*rzecz rz*rzecz )
    SWAP  ;     (  rz*rzecz ur*rzecz )
```

```
:  X*       ( r1 u1 r2 u2 --- r1.r2-u1.u2 r1.u2+r2.u1 )
    2OVER 2OVER   ( C1  C2 C1  C2 )
    ROT *         ( C1  C2 r1  r2 u1.u2 )
    >R * R> -     ( C1  C2 r1.r2-u1.u2 )
    >R            ( r1 u1 r2 u2 )
ROT ROT *         ( r1 u2 u1.r2 )
>R * R> +         ( r1.u2+u1.r2 )
R> SWHP ;         ( r1.r2-u1.u2 r1.u2+u1.r2 )
```

12.5 Funkcje trygonometryczne

Problem polega na stworzeniu funkcji trygonometrycznych w postaci całkowitoliczbowej. Ponieważ są to funkcje ciągłe, sprzężone i okresowe, łatwo jest generować tablicę ich wartości. Należy określić wielkość kroku stosownie do wymaganej dokładności, czyli liczbę cyfr po kropce. Tu przyjęto, że będzie ona wynosiła do czterech cyfr.

```
-+TABLICPA  <BUILDS  0  DO  ,     LOOP DOES>
            SWAP  2  *  +  @ ;
```

Słowo TABLICA tworzy i wypełnia tablicę, której liczba wejść podana jest jako parametr.

```
91 TABLICA SINTABLICA
```

Następne słowa umożliwiają dostęp do wartości dla wszelkich kątów.

```
: S180  DUP  90  >  IF  180 SWAP  -  THEN SINTABLICA  ;
: SIN   ( arg --- sin(arg) )
    360  MOD DUP  0<  IF  360  +  THEN  DUP  180  >
    IF  180  -  S180  MINUS  ELSE  S180  THEN  ;
: COS  90  +  SIN  ;
: TAN  DUP  SIN  SWAP  COS  DUP  ABS  2920  >
    IF  10000  SWAP */
    ELSE  DROP  DROP  ." ZR DUZE! "  THEN  ;
```

12.6 Sortowanie

Często zachodzi potrzeba uporządkowania danych, a w szczególności ułożenia liczb w rosnącej lab malejącej kolejności. Zapotrzebowanie na algorytmy zapewniające efektywne i szybkie sortowanie doprowadziło do powstania ich kilku podstawowych typów. Ich opis zawiera m.in. praca [16].
Punktem wyjścia do sortowania liczb jest stworzenie tablicy, do której moglibyśmy wprowadzać wartości wymagające posortowania. Ponieważ chodzi tu jedynie o zademonstrowanie metod, przyjmiemy, że nasza tablica będzie zawierać tylko pięć wartości. Oczywiście, nic prócz miejsca w pamięci nie stoi na przeszkodzie temu, by sortować w taki sam sposób setki i tysiące liczb.

Tablicę LICZBY tworzymy posługując się poznanym już słowem definiującym:

```
ARRAY <BUILDS 2 * ALLOT DOES> SWAP 2 * + ;
```

Zdefiniujemy ponadto zmienną NAST do przechowywania wartości podczas sortowania oraz słowa: WSTAW, do umieszczenia w naszej tablicy na próbę kilku wartości, POKAZ do ukazania zawartości tablicy i POROWNAJ do porównywania liczb. Słowo POKAZ nie jest konieczne, ale pozwoli nam obserwować sortowanie.

```
0 VARIABLE NAST
: WSTAW 120 23 157 18 93
5 0 DO I LICZBY ! LOOP ;
; POKAZ 5 0 DO I LICZBY ? LOOP ;
```

POROWNAJ oczekuje na stosie numerów porównywanych liczb (a nie ich wartości) , przy czym numer niższej wartości powinien być wprowadzony jako pierwszy.

```
: POROWNAJ      ( n1 n2 --- )
    DUP         ( n1 n2 n2 )
    LICZBY @    ( n1 n2  L2 )
    ROT DUP     ( n2 L2 n1 n1 )
```

```
LICZBY @      ( n2 L2 n1 L1 )
ROT           ( n2 n1 L1 L2 )
OVER OVER     ( n2 n1 L1 L2 L1 L2 )
        >     ( n2 n1 L1 L2 f )
IF            (  n2 n1 L1 L2 )
  ROT         ( n2 L1 L2 n1 )
  LICZBY  !   ( n2 Li )
  SWAP        ( L1 n2 )
  LICZBY !    ( --- )
ELSE 2DROP 2DROP
THEN  ;
```

Prześledzenie kolejnych faz wykonania słowa POROWNAJ pozwala zrozumieć mechanizm zamiany dwóch liczb w komórkach tablicy, gdy mniejsza liczba znajduje się dalej niż większa. Jeżeli kolejność liczb jest właściwa, POROWNAJ kończy działanie usunąwszy przedtem ze stosu liczby, które nie są już potrzebne.

12.6.1 Sortowanie przez przestawianie

Z pomocą słowa POROWNAJ możemy zorganizować sortowanie według metody noszącej nazwę: sortowanie przez przestawianie (ang. exchange sort) . Polega ono na tym, że dokonuje się przestawień aż do całkowitego uporządkowania zbioru. Proces dzieli się na następujące kroki:

1. Porównujemy pierwszą liczbę z drugą i jeżeli jest większa, zamieniamy, a jeżeli jest mniejsza, nie dokonujemy zamiany. Potem kolejno aktualnie pierwszą liczbę porównujemy z trzecią, czwartą itd. aż do końca tablicy.
2. Powtarzamy czynność porównując drugą liczbę z trzecią i następnymi.
3. Porównujemy z następnymi trzecią, czwartą i dalsze liczby.

Wstawmy teraz nasze liczby do tablicy i sprawdźmy, czy tam są:

```
WSTAW                /Return/ ok
POKAZ /Return/ 120 23 157 18 93 ok
```

Potrzebne jest teraz słowo, które będzie wykonywać z pomocą POROWNAJ operację opisaną w punkcie 1.

```
: PRZEJRZYJ  DUP  1+  NAST !
  BEGIN  OVER  1+  NAST  @  >
  WHILE  DUP  NAST  @  POROWNAJ  NAST  @  1+  NAST !
  REPEAT  2DROP  ;
```

PRZEJRZYJ posługuje się pomocniczą zmienną NAST, do której wstawia kolejne porównywane liczby. W ten sposób pisząc

```
4  0  PRZEJRZYJ
```

Spowodujemy, że PRZEJRZYJ porówna zawartość komórki zerowej, czyli 120, z zawartością kolejnych komórek aż do czwartej, czyli ostatniej, i wstawi w drodze zamiany do komórki 0 najmniejszą z napotkanych liczb, czyli 18. Sprawdźmy:

```
POKAZ  (Return) 18 23 157 120 93 ok
```

Stosując 4 1 PRZEJRZYJ, 4 2 PRZEJRZYJ itd możemy uporządkować tablicę. Uczyni to za nas słowo EXSORT.

```
: EXSORT  BEGIN  OVER  OVER
    PRZEJRZYJ  1+  OVER  OVER  =  UNTIL  2DROP  ;
```

Wstawmy ponownie z pomocą WSTAW wartości do tablicy 1, sprawdźmy działanie.

```
WSTAW  4  0  EXPORT  CR  POKAZ  /Return/
18 23 93 120 157 ok.
```

Tablica została uporządkowana.

12.6.2 Sortowanie pęcherzyków

Trochę zabawna nazwa kolejnej metody sortowania wynika z chęci obrazowego przedstawienia faktu, iż mniejsze liczby niby pęcherzyki powietrza w wodzie unoszą się w stronę początku tablicy. Zasadą sortowania pęcherzykowego (ang. bubble sort) jest kolejne porównywanie sąsiednich liczb i ich zamiana w przypadku, gdy większa poprzedza mniejszą. Proces ten powtarzany jest wzdłuż całej tablicy aż do całkowitego jej uporządkowania.

W celu stwierdzenia, czy takie uporządkowanie już nastąpiło, niezbędny jest znacznik w postaci zmiennej:

```
0 VARIABLE KONIEC?
```

Zmienna KONIEC? otrzymuje przed każdym przejrzeniem tablicy wartość 1. Każda zamiana liczb powoduje, że KONIEC? przybiera wartość 0. Jeżeli po zakończeniu kolejnego przejścia tablicy KONIEC? zachowa wartość 1, oznaczać to będzie, iż żadna zamiana nie była dokonana i że jest to koniec sortowania.

Wprowadzenie tej kontrolującej zmiennej wymaga drobnej zmiany w słowie POROWNAJ, a także zdefiniowania słowa PRZESUN, które wykona przesuwanie mniejszych liczb w lewo. Całość sortowania pęcherzykowego wykona słowo BUBSORT. Oto definicje:

```
: PORBUB
      DUP  LICZBY  @  ROT  DUP  LICZBY  @  ROT  OVER  OVER
      >  IF  ROT  LICZBY  !  SWAP  LICZBY  !  0  KONIEC?  !
      ELSE  2DROP  2DROP  THEN  ;
: PRZESUN
      BEGIN  DUP  DUP  1+  PORBUB  1+  OVER  OVER  =
      UNTIL  DROP  DROP  ;
:  BUBSORT
      BEGIN  1  KONIEC?  !  OVER  OVER  PRZESUN  KONIEC?  @
      UNTIL  2DROP  ;
```

12.6.3 Sortowanie szybkie

Dwa omówione typy sortowanie przydatne są w przypadku, gdy liczba sortowanych elementów jest nieduża. Przy jej zwiększaniu się czas wykonania szybko rośnie. Dwukrotnie więcej liczb wymaga czterokrotnie większego czasu sortowania. Na tempo sortowania pęcherzykowego wpływa także rozmieszczenie liczb. Gdy małe wartości są przy końcu tablicy, potrzeba znacznie więcej przestawieni. W obu typach sortowania niska efektywność jest przede wszystkim następstwem małej wydajności przestawień.

Sortowanie szybkie (ang. quick-sort) w poważnej mierze usprawnia proces sortowania. Opiera się ono na innych rozwiązaniach niż dotychczas omówione.

Wybieramy wartość znajdującą się blisko środka tablicy zwaną komparandem. Ustanawiamy dwa wskaźniki: lewy LW i prawy PW, którym najpierw przypisujemy krańcowe pozycje tablicy. W naszym przypadku LW=0,PW=4. Zwiększamy LW do chwili, gdy wskaże komórkę zawierającą liczbę większą lub równą wartości środkowej. W naszym przypadku LW dojdzie do wartości środkowej 157. Następnie zmniejszamy PW do chwili, gdy wskaże wartość mniejszą lub równą wartości środkowej. W naszym przypadku PW od razu wskazuje taką wartość, ponieważ liczba 93 na prawym końcu tablicy jest mniejsza od 157. W tym wypadku dokonywana jest zamiana miejscami tych dwóch liczb.

Mechanizm sortowania szybkiego polega na takim manewrowaniu obu wskaźnikami, które wywołuje kolejne zamiany wartości stojących w niewłaściwych miejscach. Powoduje to, że tablica porządkowana jest od obu jej końców.

Niezbędne jest stworzenie czterech zmiennych zawierających krańcowo adresy sortowanego obszaru oraz oba wskaźniki:

```
0 VARIABLE POCZ 0 VARIABLE KON 0 VARIABLE LW 0 VARIABLE PW
```

Słowa LEWY I PRAWY wykonują manewry wskaźnikami. CZYTAJ używane jest do odczytywania wartości z wskazanych komórek tablicy, a WSTAW - do umieszczania w nich wartości. Np. LW CZYTAJ odczyta wartość z komórki wskazanej przez lewy wskaźnik, a PW WSTAW wprowadza wartość znajdującą się jako druga na stosie do komórki wskazanej przez prawy wskaźnik.

Słowo ZAM nie wymaga do wykonania zmiany żadnych wartości na stosie. Odczytuje ono te wartości z pomocą przed chwilą omówionych słów. SORTUJ wykonuje zasadnicze czynności sortowania. Oczekuje ono na stosie wartości komparanda.
Jego wartość oblicza słowo KOMPARAND na podstawie podanych zmiennych POCZ i KON.

Posługując się tymi słowami, których funkcjonowanie proponujemy szczegółowo zanalizować, QUICK realizuje sortowanie. Jest to procedura rekurencyjna.

W celu umieszczenia w zmiennych krańcowych numerów tablicy definiujemy główne słowo QUICKSORT. Oczekuje ono tych właśnie numerów. Oto komplet omówionych definicji:

```
: LEWY  BEGIN  DUP  LW  @  LICZBY  @  >  WHILE  LW  @
        1+  LW  !  REPEAT  DROP  ;
: PRAWY  BEGIN  DUP  PW  @  LICZBY  @  <  WHILE  PW  @
        1 -  PW  !  REPEAT  DROP  ;
: CZYTAJ  @  LICZBY  @  ;
: WSTAW  @  LICZBY  !  ;
: ZAM  LW  CZYTAJ  PW  CZYTAJ  LW  WSTAW  PW  WSTAW
        LW  @   1+   LW  !  PW  @  1 - PW  !   ;
: SORTUJ  BEGIN  DUP  DUP  LEWY  PRAWY  LW  @  PW  @  >
        DUP  IF  ELSE  ZAM  THEN  UNTIL  DROP  ;
: KOMPARAND  OVER  OVER  LW  !  PW  !  + 2 / LICZBY @ ;
: QUICK  [  SMUDGE ]  KOMPARAND  SORTUJ
        POCZ @  PW @ <
        IF  PW  @  DUP  KON  !  POCZ  @  QUICK  THEN
        LW  @  KON  @  <
        IF  KON  @  LW  @   DUP  POCZ  !  OUICK  THEN
        [ SMUDGE ]  ;
: OUICKSORT  OVER  OVER  POCZ  !   KON  !  QUICK  ;
```

A N E K S Y

ANEKS 1
BIBLIOGRAFIA

Publikacje dotyczące FORTH-a

1. Moore C.H. FORTH: A New Way to Program Mini-Computers. Astronomy an Astrophysics, Suppl. 15, 1974
2. Armstrong M.A. Learning FORTH. A Self-Teaching Guide. J. Wiley and Sons Inc. 1985
3. Bishop 0. we Współpracy z Bishop A. Exploring FORTH. Granada Publishing 1984
4. Emery G. The Students' FORTH. Blackwell Scientific Publications 1985
5. Salman W.P., Tisserand O., Toulout B. FORTH. Macmillan Publishers Ltd. 1984 .
6. Winfield A. The Complete FORTH. A New Way to Program Microcomputers. Sigma Technical Press 1985
7. --- Extended fig-FORTH. Instrukcja i listingi, b.d.
8. --- Initiation au langage FORTH. Cykl artykułów w miesięczniku Microsystemes nr 35-39, /10.1983-2.1984/
9. Chrotko G. Jazyki programmirowanija wysokogo urowmia dla mikro-EWM. Moskwa 1985
10. Helms H.L. Jazyki programmirowanija. Kratkoje rukowodstwo. /Compute Language. Reference Guide. Second Edition/. Moskwa 1985

Inne publikacje

11. Dijkstra E.W. Umiejętność programowania /A Discipline of Programming/, Warszawa 1985 WNT
12. Myers G.J. Projektowanie niezawodnego oprogramowania. / Sofware Relibility. Principles an Praccitices / . Warszawa 1980 WNT
13. Turski W.M. Metodologia programowania. Warszawa 1982 WNT
14. Turski W.M. Propedeutyka Informatyki. Warszawa 1985 WNT
15. Chadwick I. Mapping the Atari. Compute! Books 1985
16. Reingold E.M., Nievergelt J., Deo N. Algorytmy kombinatoryczne. Warszawa 1985 PWN
17. Fox J.M. Programmnoje obiespieczenije i jego razrabotka / Software and Its Development /. Moskwa 1965
16. praca zbiorowa Architiektura i programmnoje osnaszczenije cyfrowych sistiem. Moskwa 1984
19. praca zbiorowa Diałogowyje mikrokompiutiernyje sistiemy. Moskwa 1986

ANEKS 2
PEŁNY EDYTOR FORTH-a

Przedstawiony tu edytor jest jedną, a licznych wersji edytorów FORTH-a. Został on opublikowany przez FORTH Interest Group w "Installation Manual" 8/80 WFR. Przedstawili go w swej książce Salman, Tisserand i Toulout [5] . "W książce tej znajduje się również pełny opis i tekst źródłowy innego edytora, opracowanego przez S.H. Daniela dla FORTH, Inc. i ogłoszonego w publikacji F.I.G. "Forth Dimensions".

Kody źródłowe obu edytorów znajdują się. na dyskietce wersji Extended fig-FORTH oraz w instrukcji do tej wersji [7].

Rozwiązania zastosowane w obu edytorach w pewnej mierze pokrywają się. Edytor S.H. Daniela jest nieco bogatszy w środki, równocześnie jednak bardziej skomplikowany. Dlatego do przedstawienia i omówienia wybrałem edytor fig-FORTH-a z uwzględnieniem zmian dokonanych przez P. Mullarky'ego. Opis funkcjonowania edytora i jego poszczególnych słów opracowany został z wykorzystaniem obu wspomnianych publikacji.

A.2.1 Słownik edytora

Edytor tu omawiany przystosowany został do pracy z 1024-bajtowymi ekranami FORTH-a zorganizowanymi w bloki po 128 bajtów podzielonymi na 16 linii, z których każda liczy 64 znaki. Jest to zatem standard fig-FORTH-a. Zaletą edytora jest jego prostota i łatwość obsługi. Zapewniają ją omówione poniżej słowa. Ponadto edytor zawiera szereg definicji słów pomocniczych" których znaczenie omówione zostało w komentarzach do tekstu źródłowego.

L / --- / Listuje bieżący ekran wywołany przez LIST.

T / n --- / Drukuje linię n i umieszcza kursor na jej Początku

E /n --- / Kasuje linię n.

D /n --- / Usuwa linię n, podciąga wszystkie następne, a tekst linii przenosi do buforu, tak iż może być ona wstawiona gdzie indziej z pomocą I.

P /n --- / Umieszcza tekst, który napiszemy, w linii n kasując jaj poprzednią treść. Służy do tworzenia nowych linii przenosząc do nich najwyżej po 64 znaki.

I /n --- / Wstawia linię z tekstem przeniesionym uprzednio do buforu z pomocą D. Obniża dotychczasowe linie od n włącznie gubiąc ostatnią.

F / --- cccc / Znajduje łańcuch cccc na bieżącym ekranie rozpoczynając od bieżącej pozycji kursora.

B / --- / Cofa kursor do początku wyrazu znalezionego z pomocą F.

C / --- cccc / Wprowadza znaki cccc do bieżącej linii od bieżącej pozycji kursora przesuwając resztę tekstu.
Jeżeli linia przekroczy 64 znaki, jej końcowa część będzie utracona.

M /n --- / Przemieszcza kursor o n znaków naprzód lub wstecz, jeżeli n jest ujemne.

S /n --- / Rozsuwa bieżący ekran tworząc nową linię bezpośrednio nad linią n i obniżając pozostałe. Ostatnią gubi.

X / --- cccc / Wyciąga łańcuch cccc i skraca linię. Jest to komenda typu "znajdź i skasuj", find-and-delete.

H / n --- / Kopiuje linię n do buforu. Tekst jest gotów do wyświetlenia.

CLEAR /n --- / Oczyszcza ekran n wypełniając go w całości spacjami. Niszczy informacje, które zawierał ekran.

COPY/n m --- / Kopiuje ekran n na ekran m. Niszczy informacje na ekranie m.

MARK / --- / Zaznacza bieżący ekran jako zmieniony. Gdy następnie użyjemy FLUSH, cały ekran zostanie wpisany na dyskietkę. Można wykorzystać do kopiowania pojedynczych ekranów na inną dyskietkę.

WHERE / --- / Używane, gdy przy ładowaniu ekranu sygnalizowany jest błąd kompilacji. Podaje numer ekranu i wskazuje strzałką miejsce błędu. Czyni ekran z błędem ekranem bieżącym, co pozwala wywołać go z pomocą L i poprawić.

Edytor obok słów, które omówiliśmy w punkcie 11.2, wykorzystuje również podstawowe słowa, które niekiedy znajdują się w słowniku predefiniowanym. Są to:

TEXT /c --- / Przesuwa linię tekstu ze strumienia prowadzania do buforu pod PAD aż do napotkania znaku ograniczającego c.

LINE / n---adr / Podaje adres początku linii n w buforze ekranu. Wgrywa blok, jeżeli nie znajduje się on jeszcze w buforze stacji dyskietek.

-TEXT /adr1 n adr2 --- f/
Jeżeli łańcuch n znaków spod adr1 występuje pod adresem adr2, zostawia tf. W przeciwnym razie ff.

MATCH /adr1 n1 adr2 n2 --- f n3/
Porównuje n1 bajtów poczynając od adr1 z zawartością bloku n2 bajtów od adr2 i jeżeli znajdzie takich samych n1 bajtów, podaje ich adres na szczyt stosu oraz znacznik logiczny 1. W przeciwnym wypadku tylko ff=0. Wykorzystywane w szeregu definicji edytora.

A.2.2 Listing edytora

Pierwsze słowa wykorzystywane przez edytor wprowadzane są do podsłownika FORTH.

```
        FORTH DEFINITIONS HEX
: TEXT
        HERE     / adres początku buforu słów, w którym
                   interpretator tekstu umieszcza znaki /
        C/L 1+   / liczba znaków w linii plus 1 /
        BLANKS   / oczyszcza /
        WORD     / przesuwa tekst ograniczony przez c ze
        HERE     / strumienia wprowadzania do buforu słów /
        PAD      / adres buforu tekstów /
        C/L 1 +
        CMOVE ;  / przesuwa tekst 64 bajtów+długość, pod PAD /
:LINE
        DUP FFF0 AND

        17 ?ERROR /upewnia się, czy numery linii mieszczą się /
                  / w granicach od zera do 15 /
        SCR @     / pobiera numer ekranu /
        (LINE)    / przenosi ekran z buforu stacji dyskietek do
                    buforu ekranu, ewentualnie ładując blok do
                    buforu stacji dyskietek. Oblicza adres n-tej
                    linii i umieszcza na stosie /
        DROP ;

: MARK 10 0 DO I LINE UPDATE DROP LOOP ;
: -TEXT SWAP -DUP IF OVER + SWAP DO DUP C@ I C@ - IF 0=
        LEAVE ELSE 1+ THEN LOOP ELSE DROP 0= THEN ;
: MATCH >R >R 2DUP R> R> 2SWAP OVER + SWAP
        DO 2DUP FORTH I -TEXT IF >R 2DROP  R>
        - I SWAP - 0 SWAP 0 0 LEAVE THEN LOOP
        2DROP SWAP 0= SWAP   ;
```

Następne definicje wprowadzane są do podsłownika EDITOR.

```
VOCABULARY  EDITOR  IMMEDIATE HEX  EDITOR DEFINITIONS
```

```
: WHERE DUP B/SCR / DUP SCR !  ." SCR # " DECIMAL
        .  SWAP C/L /MOD C/L * ROT BLOCK + CR
        C/L TYPE CR HERE C@ - SPACES 5E EMIT
         [COMPILE]  EDITOR QUIT ;
EDITOR DEFINITIONS
: -MOVE           / adr n --- kopiuje linię tekstu od adr w
                    n-tej linii bieżącego buforu ekranu /
        LINE      / oblicza adres linii n /
        C/L CMOVE / przemieszcza 64 znaki spod adresu do linii
                    n /
        UPDATE ;  / informuje manipulator stacji dyskietek, że
                    blok został zmieniony /
: H               / n---/
        LINE      / oblicza adres linii /
        PAD 1+    / adres początkowy PAD /
        C/L DUP   / liczba znaków w linii /
        PAD C!    / ustala długość PAD na 64 /
        CMOVE ;   / przemieszcza n-tą linię /
: E               / n --- /
        LINE       / adres linii /
        C/L BLANKS UPDATE ; / wypełnienie spacjami /
: S               / n---/
        DUP 1 -   / linia do przesunięcia /
        OE        / 14, numer ostatniej linii do przesunięcia/
        DO I LINE / oblicza adres n-tej linii /
        I 1+      / numer następnej linii /
        -MOVE     / przesuwa linię w dół /
        -1 +LOOP  / zmniejsza wskaźnik pętli o 1 /
        E ;       / kasuje n-tą linię /
```

```
: D               / n --- /
        DUP H     /kopiuje n-tą linię do PAD /
        0F        / 15, numer ostatniej linii /
        DUP ROT DO
        I 1+ LINE / adres linii do przesunięcia /
        I -MOVE   / przesuwa linię w górę /
LOOP E ;          / kasuje ostatnią linię /
: R               / n --- /
        PAD 1+    / adres początku PAD /
        SWAP -MOVE ; / kopiuje tekst z PAD do n-tej linii /
: P               / n --- /
        1 TEXT R ; / wprowadza tekst zakończony CR do n-tej
                     linii /
: I               / n --- /
        DUP S     / oczyszcza n-tą linię /
        R ;       / kopiuje tekst z PAD do n-tej linii /
```

Następne dwa słowa działają na całym ekranie:

```
: CLEAR           / n --- /
        SCR !     / określa numer ekranu /
        10 0 DO   / 16 dec 0 - granice pętli /
        FORTH I   / informuje, że chodzi o I.FORTH-a /
        EDITOR E  / powrót do edytora i skasowanie linii I /
        LOOP ;    / skasowanie 15 linii /
: COPY            / n m --- /
        B/SCR *   / pierwszy blok odpowiadający ekranowi m /
        OFFSET @ + / adres fizyczny na dyskietce /
        SWAP      / adr n /
        B/SCR *   / pierwszy blok odpowiadający ekranowi n /
```

```
        B/SCR OVER + / adr blok2 blok2+1 /
        SWAP        / adr /m/ koniec/n/+1 początek/n/+1 /
        DO          / adr/m/ /
        DUP FORTH I
        BLOCK       / ładuje bieżący blok ekranu n do buforu i
                      podaje jego adres /
        2 - !       / podaje jako jego numer - numer bieżącego
                      bloku ekranu m /
        1 +         / przemieszcza następny blok na ekran m /
        UPDATE      /sygnalizuje, że blok uległ zmianie /
        LOOP DROP FLUSH ; / przenosi ekran na dyskietkę /
```

Wszystkie dotychczas zdefiniowane słowa składają się na nieduży edytor liniowo-stronicowy w tym znaczeniu, że podstawową jednostką, którą można zmienić, jest pełna linia tekstu.

Następne słowa rozszerzają możliwości: pozwalają dokonywać zmian w samej linii. Wykorzystuje się tu zmienną R# zawierającą pozycję kursora wskazującego następny znak w linii. Pierwsze cztery słowa mają charakter pomocniczy i służą do zdefiniowania następnych.

```
: TOP 0 R#  ! ;    / umieszcza kursor na początku ekranu /
: #LOCATE          / --- ni n2 oblicza numer linii i przesunię-
                     cie kursora /
    R# @ C/L /MOD ;
: #LEAD            / --- adr n oblicza adres linii w buforze
                     ekranu adr i przesunięcie w linii /
    #LOCATE LINE SWAP ;
: #LAG             / --- adr n oblicza to samo, tyle że w
                       stosunku do końca linii /
#LEAD DUP >R + C/L R> - ;
: M                / n --- /
    H +!           / przesuwa kursor /
```

```
        CR SPACE   /nowa linia /
        #LEAD TYPE /wyświetla tekst przed kursorem /
        5E EMIT    / wyświetla kursor, znak potęgowania/
        #LAG TYPE  / wyświetla resztę linii /
        #LOCATE .  /  wyświetla numer linii /
        DROP ;
: T             / n --- /
        DUP C/L *  /oblicza miejsce linii w buforze /
        R# !       /  ustawia kursor na początku linii /
        H          / kopiuje n-tą linię do PAD /
        O M ;      / drukuje n-tą linię na urządzeniu /
: L             / --- /
        SCR @ LIST / listuje ekran /
        O M ;      / wyświetla linię zawierającą kursor /
```

Słowa 1LINE, FIND, DELETE i TILL mają w zasadzie charakter pomocniczy. Działanie pozostałych omówiliśmy.

1LINE bada, czy w linii jest ciąg znaków z PAD.

FIND poszukuje w całym ekranie ciągu znaków z PAD. Daje komunikat o błędzie lub zmienia pozycję kursora.

```
        DELETE     / n---/ kasuje n znaków od pozycji kursora i
                     koryguje linię.
```

TILL kasuje wszystkie znaki za bieżącą pozycją kursora aż do pierwszego wystąpienia tekstu, który następuje dalej w strumieniu wprowadzania.

Podajemy te definicje bez komentarzy.

```
: 1LINE  #LAG  PAD  COUNT  MATCH  R#  +!  ;
: FIND  BEGIN  3FF  R#  @  <  IF  TOP  PAD  HERE  C/L 1+  CMOVE
        0 ERROR THEN 1LINE UNTIL ;
: DELETE  >R  #LAG  +  FORTH  R  -  #LAG  R  MINUS
R#  +!  #LEAD  +  SWAP  CMOVE  R>  BLANKS  UPDATE  ;
```

```
: N     / znajduje pierwsze wystąpienie łańcucha z PAD /
        FIND O M ;
: F               / --- cccc /
        1 TEXT N ;
: B               / --- /
PAD C@            / wyświetla długość ciągu znaków w PAD/
MINUS M ;         / cofa kursor i ponownie wyświetla linię /
: X               / --- ccc /
        1 TEXT    / relokacja w PAD /
        FIND      / przeszukiwanie ekranu /
        PAD C@    / długość ciągu do skasowania /
        DELETE    / kasuje /
        O M ;     / wyświetla /
        : TILL  #LEAD  +1  TEXT  1LINE  0=  0  ?ERROR
                  #LEAD  +  SWAP  -  DELETE  O  M ;
: C
1 TEXT PAD COUNT #LAG ROT OVER MIN >R FORTH R R# +!  R  - >R
DUP HERE R CMOVE
HERE  #LEAD  +  R>  CMOVE  R>  CMOVE  UPDATE  O  M  ;
                  FORTH DEFINITIONS DECIMAL
```

ANEKS 3
FORTH NA ATARI

A.3.1 Specyficzne cechy

Implementacje FORTH-a na komputery Atari nie odbiegają w zasadniczych rozwiązaniach od tych, które zostały wcześniej przedstawione. Parokrotnie wspomnieliśmy o pewnych specyficznych rozwiązaniach, dostosowujących interpretator FOTH-a do architektury 8-bitowych komputerów Atari. W przypadku fig-FORTH-a było to szczególnie łatwe ze względu na to, że autorzy tego standardu języka opracowywali go posługując się komputerami z mikroprocesorem 6502, a więc tym, który stanowi "serce" Atari.

Najważniejsza właściwość implementacji na Atari - to umieszczenie stosu głównego na stronie zerowej, buforu klawiatury na dolnej połowie strony 1, a buforów wprowadzania z urządzeń zewnętrznych poniżej słownika. Pozostała część stosu maszynowego Atari mieszczącego się na str. 1 wykorzystana jest do budowy stosu powrotów.

Warto poświęcić kilka słów stronie zerowej. Ze względu na to, że posługiwanie się nią znacznie skraca czas wykonania, każdy bajt na niej jest szczególnie ceniony. Pierwszych 128 bajtów (0-127) wykorzystanych jest na komórki ważne dla systemu operacyjnego Atari. Następnych 128 bajtów przeznaczonych jest do wykorzystania przez języki programowania, przy czym górna część tego obszaru 210-255 wykorzystywana jest przez programy obliczeń zmiennoprzecinkowych, gdy takie obliczenia są dokonywane.

W FORTH górną połowę strony zerowej z reguły wykorzystuje się w maksymalnym stopniu, przy czym znaczną część tego miejsca zajmuje stos główny, pozostałą zaś - komórki sterujące arytmetyką FORTH-a.

Takie usytuowanie stosu ma wielki plus w postaci przyspieszenia obliczeń. Powoduje zarazem ograniczenie długości stosu. Rosnąc w dół nie może on wkroczyć na obszar komórek sterujących wielu funkcjami komputera, bowiem powoduje to generalne "rozregulowanie" jego pracy z zawieszaniem się systemu włącznie. Stąd prosty wniosek! trzeba jak najpełniej wykorzystać walor szybkości, natomiast przez dokładne programowanie i racjonalne posługiwanie się stosem zapobiegać jego nadmiernemu rozrastaniu się, trzymać go w ryzach.

A. 3.2 EXTENDED fig-FORTH

Najciekawszą ze znanych mi implementacji FORTH-a na Atari jest wspomniany już parokrotnie Extended fig-FORTH. .Jego podstawowe cechy zbieżne są z przedstawionymi w poprzednim punkcie. Równocześnie w implementacji tej zastosowano szereg oryginalnych, często zaskakujących prostotą rozwiązań. Wersja ta jest automatycznie wgrywana do pamięci po włączeniu komputera. Nie korzysta ona z żadnego DOS.

Extended fig-FORTH ściśle przestrzega standardu fig-FORTH-a. W zestawie predefiniowanych słów stosunkowo szeroko uwzględniono słowa do operacji na liczbach podwójnych. Wprowadzono słowa SAVE i CSAVE umożliwiające ponowne zapisanie na dyskietkę lub taśmę całego interpretatora łącznie z wszystkimi słowami zdefiniowanymi przez użytkownika. Automatycznie nakładana jest przy tym ochrona przed skasowaniem na cały nowo zapisany słownik.

Umożliwia to wygodne tworzenie pakietów w postaci gotowego do wykonania kodu maszynowego. W Extended fig-FORTH zadbano o pozostawienie dostatecznie dużego miejsca dla wersji wgrywanej z pomocą SAVE. Dlatego ekrany z błędami z ich standardowej pozycji przesunięto pod numery 14 i 15. Także MESSAGE działa na tych ekranach.

Udane rozwiązanie zastosowano w dziedzinie posługiwania się klawiaturą i ekranem. W przeciwieństwie do wielu mniej udanych wersji definicje tworzone wprost z klawiatury, czyli bez użycia edytora FORTH-a, można łatwo poprawiać cofając kur-

sor do linii z błędem, poprawiając ją i kontynuując pisanie definicji. W przypadku definicji zakończonej można ją anulować z pomocą FORGET, wrócić na znajdujący się na ekranie tekst i dokonać poprawek. By ponownie zdefiniować słowo, trzeba cofnąć kursor na otwierający je dwukropek.

Udogodnieniem jest również to, że po VLIST można w każdej chwili przerwać wyświetlanie listy naciskając dowolny klawisz. Oszczędza to sporo czasu. Istotną cechą implementacji jest to, że po naciśnięciu klawisza RESET FORTH jest ponownie wywoływany. Niknie przy tym ostatnio zdefiniowana część słownika, pozostają natomiast bufory ekranów.

Na firmowej dyskietce w postaci skompilowanej znajduje się jedynie predefiniowany podsłownik FORTH-a. Równocześnie dyskietka ta zawiera bogate oprogramowanie pomocnicze, a mianowicie:

- dwa odrębne edytory

- dwa asemblery o dość odmiennych rozwiązaniach

- debugger czyli program uruchamiający

- zestawy definicji do operowania grafiką i dźwiękiem

- pakiet definicji umożliwiających pełne wykorzystanie podprogramów systemu operacyjnego ATARI

- pakiet obsługi stacji dyskietek, włącznie z formatowaniem i kopiowaniem

- pełny pakiet do przejścia w obrębie FORTH-a na arytmetykę zmiennopozycyjną.

Ponadto dołączono krótki zestaw definicji zapewniających pełną kompatybilność z wersją FORTH-a, którą posługuje się Leo Brodie w swej popularnej książce "Starting FORTH".

Jak widać, jedna strona dyskietki wystarczyła na to, by zebrać bogate oprogramowanie użytkowe, które w dowolnych zestawach można wprowadzać do słownika, a następnie tworzyć z pomocą SAVE skompilowane wersje do szybkiego wykorzystania.

Dla osób starających się poznać FORTH-a i pogłębić wiedzę o nim, listingi dostarczone przez Extended fig-FORTH są

ponadto bardzo ciekawym materiałem. Zaletą implementacji jest to, że towarzyszy jej obszerna instrukcja wyjaśniająca w szczególności zastosowane specyficzne rozwiązania i ułatwiająca posługiwanie się interpretatorem. Instrukcja zawiera ponadto wydruk listingów z wszystkich ekranów. . Patrz [7]

A.3.3 Inne narzędzia

Z Extended fig-FORTH-a zaczerpnęliśmy zestaw definicji sterujących dźwiękiem, praktycznie identycznych ze stosowanymi przez BASIC, z dodatkowym słowem FILTER!
SOUND (n1 n2 n3 n4 ---) ma kolejność wartości taką samą jak w BASIC-u, tyle że liczby należy podać przed słowem.
FILTER! (n ---) wprowadza wartości od 0 do 255 do rejestru kontroli dźwięku AUDCTL.
Oto listing :

```
BASE @ HEX

B208 CONSTANT AUDCTL

D200 CONSTANT AUDBASE

: SOUND 3 DUP 0D20F C! 232 C! SWAP 16 * + ROT

    DUP + AUDBASE + ROT OVER C! 1+ C! ;

: FILTER! AUDCTL C! ;
```

Brak miejsca nie pozwala na szersze przedstawienie programu uruchamiającego Extended fig-FORTH-a, zainteresowanych zachęcam do zapoznania się z listingiem na dyskietce i instrukcją. Zestaw procedur zajmuje 6 ekranów i obok omówionych przeze mnie wcześniej pomocy do odczytania wartości ze stosu zawiera użyteczne narzędzie umożliwiające odtworzenie budowy definicji określonego słowa. Gdy napiszemy słowo "DECOMP", a po nim nazwę interesującego nas słowa, komputer wyświetli sekwencję słów, z których jest ono zbudowane oraz adresy ewentualnych skoków. Dekomponuje Jednak tylko słowa zdefiniowane z pomocą definicji dwukropkowej, bez kodu maszynowego. Mimo tego ograniczenia DECOMP jest bardzo przydatne.

ANEKS 4
ROZWIĄZANIA ĆWICZEŃ

1. : NAPIS KOT ." . i poluje na myszy" ;

Po słowie ." należy dać dodatkową spację.

2. A B C -

   A B + C / D E + F +

   A B C + / D A + C D +

5. : 3DUP >R OVER OVER R> ROT ;

6. Jedna z możliwości:

   ROT >R ROT >R SWAP ROT >R ROT R> R> SWAP

   >R SWAP ROT R>

8. Wykorzystując jak najszerzej liczby pojedynczej długości i odpowiednie operatory oraz słowa wyprowadzające wynik można przedstawione działania wykonać, na przykład, następująco:

   a/ 32000 29000 M* D.

   b 32000 33000 U* D.

   c/ 1000000000. 25000 U/ U. DROP

   d/ 1500000335. 25000 U/ U. .

   e/ 777 DUP M* D.

   f/ 45333 DUP U* D.

9. : 0> MINUS 0< ;

10. W definicji INPUT należy usunąć końcowe DROP. Pamiętać trzeba o oznaczaniu z pomocą kropki liczb jako podwójnych.

11. : KOSTKA

   100 0 DO 6 RND 1+ . LOOP ;

ANEKS 5
KOMUNIKATY BŁĘDÓW

W podstawowej implementacji według fig-FORTH-a komunikaty błędów umieszczane są na ekranach 4 i 5 w poniżej podanej kolejności. W poszczególnych implementacjach umiejscowienie komunikatów oraz ich układ mogą wykazywać odchylenia. Obok komunikatów podano ich polskie znaczenia.

SCR# 4

0	(ERROR MESSAGES)	
1	Stack empty	Stos pusty
2	Dictionary full	Słownik pełny
3	Wrong address mode	Niewłaściwy tryb adresowania
4	Isn't unique	Nie jedyna (nazwa słowa)
5	Value error	Błąd wartości
6	Disk address error	Błędny adres na dyskietce
7	Stack full	Stos pełny
8	Disk error	Błąd stacji dyskietek (np. nie włączona)

SCR# 5

0	(ERROR MESSSAGES)	
1	Use only in definitions	Stosowane tylko w definicjach
2	Execution only	Tylko w fazie wykonywania
3	Conditionals not paired	Słowa w instrukcji warunkowej nie tworzą pary
4	Definition not finished	Definicja nie zakończona
5	In protected dictionary	W Słowniku chronionym (przed skasowaniem)
6	Use only when loading	Stosowane tylko podczas ładowania
7	Off current screen	Poza bieżącym ekranem
8	Declare VOCABULARY	Zadeklaruj podsłownik (z pomocą VOCABULARY)

linie 9-15 obu ekranów są puste.

ANEKS 6
LEKSYKON SŁÓW FORTHA

Leksykon zawiera opis znaczenia najbardziej standardowych słów FORTH-a zawartych w jego słowniku predefiniowanym. Objęto nim również wiele słów, które nie mają w standardzie FORTH-a charakteru obowiązkowych, lecz często włączane są do zasobu interpretatorów. W leksykonie uwzględniono występujące odmienności nazw słów pełniących takie same funkcje, np. -DUP i ?EUP. Podstawą leksykonu jest zasób słów fig-FORTH-a. Uwzględniono również słowa zalecane przez FORTH Standard Team.

Leksykon obejmuje słowa z głównego podsłownika FORTH. Specyficzne definicje z innych podsłowników, w tym edytora i asemblera, omówione zostały w odpowiednich rozdziałach książki i aneksach.

W pierwszej linii każdego hasła obok brzmienia nazwy słowa przedstawiono symbolicznie, jakie dane powinny one otrzymać na stosie i jaki wpływ wywiera na jego stan. Miejsce słowa oznaczono trzema myślnikami: "---". Jeżeli w linii jest tylko taki znak, oznacza to, że słowo nie oddziałuje bezpośrednio na stan stosu głównego. W zastosowanej notacji szczyt stosu znajduje się po prawej stronie.

Słowa ułożone są w porządku alfabetycznym według kodu ASCII.

Użyte symbole mają następujące znaczenie:

adr	adres w pamięci
a	bajt
c	znak kodu ASCII
d	32-bitowa całkowita liczba podwójna ze znakiem
f	znacznik logiczny
ff	znacznik logiczny "fałsz" - 0
n	16-bitowa liczba całkowita ze znakiem

u 16-bitowa liczba całkowita bez znaku
ud 32-bitowa całkowita liczba podwójna bez znaku
tf znacznik logiczny "prawda" - wartość niezerowa

! n adf ---
Wpisuje 16 bitów liczby n pod adres podany na szczycie stosu

!CSP ---
Nadaje CSP wartość wskaźnika stosu.

d1 --- d2
Przekształca kolejną cyfrę liczby podwójnej d1 w znak wyprowadzanego ciągu. Zostawia na stosie iloraz z dzielenia d2 przez BASE służący do dalszego przetwarzania. Używane między <# a #> Patrz także #S.

#> d --- adr długość
Kończy przekształcanie liczby w ciąg znaków, usuwa d. Zostawia adres łańcucha i jego długość do wykorzystania przez TYPE.

#S d1 --- 0 0
Przekształca wszystkie cyfry znaczące liczby podwójnej d1 w ciąg znaków w buforze wyprowadzania wykorzystując do tego #. Używane między <# a #>.

#TIB --- adr
Zostawia na stosie adres, pod którym podana jest Liczba znaków (długość) buforu terminalu.

' --- adr
Używane w postaci ' cccc zostawią na stosie adres Pola parametrów słowa ccc. W definicji dwukropkowej Powoduje skompilowanie adresu tego słowa jako liczby. Jeżeli słowo nie zostanie znalezione w podsłownikach CONTEXT i CURRENT, wywoła to komunikat o błędzie. Polska nazwa słowa : apostrof, angielska : tick.

(---
Używane w postaci

(cccc

rozpoczyna komentarz, którego koniec wyznacza znak) w tej samej linii. Może być stosowane w bieżącym wykonywaniu, jak i w definicji dwukropkowej. Po początkowym nawiasie konieczna jest spacja.

(+LOOP)
(.")
(;CODE)
(ABORT)
(DO)
Zastosowane w skompilowanej definicji słowa te w fazie wykonywania realizują działania słów zawartych w nawiasach wyjaśnione przy odpowiednich słowach.

(FMT) n1 --- n2
Formatuje dyskietkę w stacji numer 1 i zostawia na stosie bajt stanu DOS. Słowo wykorzystywane w procedurze formatowania przez podprogram systemu operacyjnego i jego słowo FORMAT. Nie do bezpośredniego stosowania.

(LINE) Wykorzystywane w definicji LINE edytora.

(LOOP)
(NUMBER)
Zastosowane w kompilowanej definicji słowa te w fazie wykonywania realizują działania słów zawartych w nawiasach.

(SAVE) ---
Zapisuje n bloków na dyskietkę w stacji nr 1 rozpoczynając od sektora 0 (sektora 1 w nomenklaturze DOS) . Nie powinno być używane w programach. Stosowane w EXTENDED-fig-FORTH.

* n1 n2 --- n3
Zostawia na stosie iloczyn pojedynczej długości ze znakiem dwóch liczb ze znakiem n1 i n2.

*/ n1 n2 n3 --- n4
Zostawia na stosie n4 = n1*n2/n3. W toku działań powstaje pośredni 31-bitowy iloczyn, co

zapewnia wyższą dokładność niż osiągana z pomocą sekwencji
n1 n2 * n3 /

*/MOD n1 n2 n3 --- n4 n5
To samo co */, zostawia jednak na stosie iloraz n4 i resztę z dzielenia n5.

+ n1 n2 --- n3
Zostawia na stosie pojedynczej długości sumę
n1 i n2.

+! n adr ---
Dodaje n do wartości znajdującej się pod adresem podanym na szczycie stosu.

+- n1 n2 --- n3
Nadaje n1 znak n2 i zostawia na stosie jako n3.

+LOOP n ---
Używane w definicji dwukropkowej w postaci:
DO...n +LOOP
W fazie wykonywania pętli dodaje n do jej wskaźnika aż do chwili, gdy wskaźnik ten stanie się równy granicy jej zakresu lub od niej większy. Wówczas pętla kończy działanie, a jej parametry są usuwane ze stosu. Podczas kompilacji, gdy słowo znajduje się na szczycie słownika,

+LOOP kompiluje słowo (+LOOP) i skok odgałęzienia od HERE do adresu zostawionego na stosie przez DO, czyli do początku pętli,

+ORIGIN n --- adr
Zostawia na stosie adres o n wyższy od początku słownika . Sekwencja
0 +ORIGIN
pozwala ustalić adres początku słownika.

, n ---
Kopiuje 16-bitową liczbę ze stosu do pierwszych komórek pamięci nad szczytem słownika, zwiększa-

jąc wskaźnik słownika i HERE o 2.

- n1 n2 --- n3
Zostawia na stosie różnicę n3 liczb n1 i n2.

--> ---
Kontynuuje ładowanie i interpretację ekranu przenosząc je na następny ekran na dyskietce.

-DISK adr n2 n3 f --- n4
Wykonuje czytanie z dyskietki lub zapisywanie na nią: adr jest adresem startowym w pamięci, n2 numerem sektora na dyskietce (0-719) , n3 numerem stacji dyskietek (1-4) , a znacznik logiczny f, ma wartość 1 dla czytania i 0 dla zapisywania. Po zakończeniu n4 będzie zerem, gdy wszystko przebiegło pomyślnie, lub numerem błędu DOS. Nie jest przewidziane korzystanie z tego słowa w normalnych programach w FORTH. Zwykle używa się w nich R/W

-DUP n1 --- n1 gdy n1 = 0
n1 --- n1 n1 gdy n1 nie równa się 0 Kopiuje n1 na szczyt stosu tylko wtedy, gdy ma ono wartość niezerową. Użyteczne zwłaszcza przy kopiowaniu wartości tuż przed IF, eliminuje bowiem konieczność tworzenia części ELSE, by usunąć tę wartość, gdy równa się ona zeru.

-FIND --- pfa b tf gdy znaleziono
--- ff gdy nie znaleziono. Służy do ustalania, czy następne słowo wprowadzane na szczyt słownika pod HERE odgraniczone spacją nie było już wcześniej zdefiniowane. -FIND przegląda podsłowniki CONTEXT i CURRENT. Jeżeli, znajdzie słowo, zostawia na stosie adres jego pola parametrów pfa, bajt długości nazwy b i znacznik logiczny prawda tf. W przeciwnym wypadku zostawia tylko znacznik fałsz (0) .

-TRAILING adr n1 --- adr a2
Zmienia liczbę znaków n1 ciągu tekstowego znajdującego się pod adresem, by zapobiec wyprowadzaniu kończących go spacji. Innymi słowy znaki od adr+n2 do adr+n1 są spacjami.

. n ---
Kropka jest słowem powodującym wydrukowanie liczby 16-bitowej ze znakiem w kodzie uzupełnieniowym i aktualnym układzie liczbowym; dziesiątkowym, szesnastkowym, dwójkowym lub innym określonym w BASE. Po liczbie drukowana jest spacja, a liczba jest usuwana ze stosu.

." Używane w postaci:

```
." cccc"
```

Kompiluje łańcuch znaków cc aż do końcowego cudzysłowu i przekazuje go następnie w fazie wykonywania procedury na wskazane urządzenie: ekran, drukarkę i in. Poza definicją powoduje natychmiastowe wyświetlenie tekstu aż do końcowego ". Maksymalna liczba znaków może zależeć od implementacji.

.LINE n1 n2 ---
Drukuje na ekranie terminalu linię o numerze n1 z ekranu na dyskietce o numerze n2. Usuwa zbędne spacje.

.R n1 n2 .R
Drukuje liczbę n1 w polu o szerokości n2 dosuniętą do prawej krawędzi pola. Nie dodaje spacji. Umożliwia sformatowane wyprowadzanie liczb.

/ n1 n2 --- n3
Wykonuje całkowitoliczbowe dzielenie n1 przez n2 i zostawia na stosie iloraz n3.

/MOD n1 n2 --- reszta iloraz
To samo co /, lecz zostawia na stosie poniżej ilorazu resztę ze znakiem dzielnej.

0, 1, 2, 3 Liczby skompilowane w słowniku, co usprawnia obliczenia.

0< n --- f
Zostawia na stosie znacznik logiczny "prawda" tf, gdy n jest ujemne lub "fałsz" w przeciwnym wypadku.

0= n --- t
Zostawia na stosie tf, gdy liczba jest równa zeru lub ff w przeciwnym wypadku.

OBRANCH --- adres względny
Procedura systemowa stosowana przy kompilacji odgałęzień, które wykonuje, gdy napotka wartość zero. Wykonuje przejście pod adres względny, tzn. obliczony w stosunku do adresu OBRANCH.

1+ n --- n+1
Zwiększa n o 1. To samo co 1 + .

2+ n --- n+2
Zwiększa n o 2.

2@ To samo co D@ .

2DROP d ---
Usuwa liczbę podwójną d ze stosu.

2DUP d --- d d
Kopiuje d na szczyt stosu.

2OVER d1 d2 --- d1 d2 d1
Kopiuje d1 na szczyt stosu.

2ROT d1 d2 d3 --- d2 d3 d1
Przesuwa d1 na szczyt stosu.

2SWAP d1 d2 --- d2 d1
Zamienia miejscami dwie górne liczby podwójne.

: ---
Używane w celu rozpoczęcia tzw. Definicji dwukropkowej o postaci:

```
: cccc ... ;
```

Tworzy nowe wejście do słownika, którego nazwą jest cccc i które jest równoważne sekwencji słów FORTH-a następującej po cccc i zakończonej średnikiem ; lub słowem ; CODE. Podczas kompilacji dwukropek sprawdza, czy STATE jest wartością niezerową, i gdy tak jest tworzy definicję nowego słowa cccc wprowadzając do niej adres odpowiednich pól słów włączonych. Są to w większości przypadków adresy pól parametrów tych słów. Po : niezbędna jest spacja.

; ---

Zamyka definicję dwukropkową kompilując jako ostatnie w niej słowo ;S, po czym kończy kompilację. Przed ; niezbędna spacja.

;CODE ---

Służy do zakończenia kompilowanej definicji dwukropkowej przy jednoczesnym dołączeniu do niej części w kodzie maszynowym, która jest następnie wykonywana przy każdym wykonaniu tego słowa, a także słów, w których skład ono wchodzi.

;S ---

Umieszczone na zakończenie ekranu na dyskietkę zatrzymuje jego interpretację. Może być także użyte w obrębie słowa, np. do sterowania rozgałęzieniami warunkowymi.

< n1 n2 --- f

Zostawia na stosie tf, jeżeli n1 jest mniejsze od n2, lub ff w przeciwnym wypadku.

<# Inicjuje wyprowadzanie liczby podwójnej w postaci przekształconej w łańcuch znaków złożonych z cyfr stosownie do aktualnej BASE uzupełnionych o inne znaki, np. kropkę lub przecinek przed ułamkiem. Ta forma konwersji liczb umożliwia przedstawianie stosowanych w FORTH działań na liczbach całkowitych i ich wyników w postaci liczb z częścią ułamkową.

Do takiej konwersji liczb służą słowa:

```
<# # #S  SIGN #>
```

Powstający w efekcie tekst umieszczany jest pod adresem PAD powyżej szczytu słownika.

<BUILDS Używane w fig-FORTH w definicji dwukropkowej w postaci:

```
: cccc <BUILDS ... DOES> ... ;
```

Za każdym razem, gdy wykonywane jest cccc, <BUILDS definiuje nowe słowo z procedurą wykonywaną na wyższym poziomie. Wykonując cccc w formie

```
cccc nnnn
```

wykorzystujemy <BUILDS do stworzenia wejścia słownikowego dla nnnn powodującego wywołanie przez nnnn części znajdującej się za DOES>. Gdy nnnn jest później wykonywane, adres tego pola parametrów wprowadzany jest na stos, a nnnn wykonuje słowa znajdujące się w cccc za DOES>. <BUILDS i DOES> umożliwiają tworzenie procedur działających w fazie wykonywania bez konieczności stosowania kodu asemblera, jak tego wymaga ;CODE. W niektórych implementacjach zamiast <BUILDS używa się CREATE.

= n1 n2 --- f
Zostawia na stosie tf, jeżeli n1 równe jest n2, w przeciwnym wypadku zostawia ff.

> n1 n2 --- f
Zostawia na stosie tf, gdy n1 jest większe od n2, w przeciwnym wypadku zostawia ff.

>IN To samo co IN.

>R n ---
Usuwa liczbę ze stosu głównego i umieszcza ją jako najbardziej dostępną na stosie powrotów.

? adr ---
Drukuje liczbę spod adresu w układzie liczbowym określonym przez aktualną BASE.

?CSP ---
Sprawdza, czy wskaźnik stosu ma w danej chwili tę samą wartość, co poprzednio wprowadzona z pomocą ! CSP. Jeżeli tak nie jest, powstaje błąd.

?COMP ---
Daje komunikat o błędzie, jeżeli STATE jest równe 0, tzn. jeżeli system nie znajduje się w fazie kompilowania.

?DUP To samo co -DUP.

?ERROR f n ---
Daje komunikat o błędzie numer n, jeżeli znacznik logiczny jest tf .

?EXEC ---
Daje komunikat o błędzie, jeżeli system nie jest w fazie wykonywania.

?LOADING ---
Daje komunikat o błędzie, jeżeli interpretowany strumień danych wprowadzany jest z terminalu (klawiatury) , a nie z dyskietki lub taśmy.

?PAIRS n1 n2 ---
Daje komunikat o błędzie, jeżeli dwie górne wartości na stosie nie są sobie równe.

?STACK ---
Daje komunikat o błędzie, jeżeli stos jest przepełniony.

?TERMINAL --- f
Sprawdza, czy terminal występuje o przerwanie. Znacznik tf wskazuje, że tak jest.

@ adr --- n
Zastępuje na stosie adres przez jogo 16-bitową zawartość.

ABORT ---

Powoduje ponowny gorący start, przy czym: zeruje Stosy: przywraca stan wykonywania; przekazuje sterowanie terminalowi: drukuje nagłówek, jaki pojawia się przy pierwszym uruchomieniu FORTH-a. Nie niszczy zmiennych użytkownika ani zawartości buforów.

ABORT" --- f

Czyni to, co ABORT, jeżeli na stosie jest ff (0)

ABS n --- u

Zastępuje liczbę jej wartością bezwzględną.

AGAIN adr n --- kompilacja

Używane w definicji dwukropkowej w postaci:

```
BEGIN ... AGAIN
```

W fazie wykonywania słowa AGAIN wymusza powrót do odpowiedniego BEGIN. Nie ma to żadnego wpływu na stos. Wykonanie nie może opuścić tej pętli (chyba że stopień niżej wykona się R> DROP). W fazie kompilacji AGAIN kompiluje BRANCH z przesunięciem od HERE do adresu. Liczbę n wykorzystuje się w fazie kompilacji do kontroli błędów.

ALLOT n ---

Dodaje liczbę n ze znakiem do wskaźnika stosu DP. Używane do zarezerwowania miejsca w słowniku, np. na tablicę, a także do odświeżenia pamięci.

AND n1 n2 --- n3

Zostawia na stosie jako n3 wynik logiczny AND na bitach liczb n1 i n2.

ASCII --- b

Daje na stos kod najbliższego drukowanego znaku ze strumienia wprowadzania.

ASSEMBLER ---
Czyni podsłownik asemblera podsłownikiem CONTEXT, przeszukiwanym w pierwszej kolejności,

B/BUF --- n
Podaje długość buforu stacji dyskietek w bajtach.

B/SCR --- n
Podaje liczbę sektorów w ekranie.

B? --- n
Podaje aktualną wartość BASE.

BACK ---
Kompiluje przesunięcie dla odgałęzień wstecz.

BASE --- adr
Zmienna użytkownika. Podaje adres zawierający bieżącą podstawę układu liczbowego stosowanego dla konwersji liczb binarnych przy wprowadzaniu i wyprowadzaniu danych. Np., by posługiwać się przy operowaniu danymi liczbowymi układem ósemkowym, należy pod adres BASE wprowadzić 8, czyli napisać:

```
8 BASE !
```

BEEP ---
Brzęczyk, sygnał dźwiękowy.

BEGIN --- adr n kompilacja
Stosowane w definicji dwukropkowej jako miejsce początku pętli nieokreślonej w konstrukcjach:

```
BEGIN ... UNTIL

BEGIN ... AGAIN

BEGIN ... WHILE ... REPEAT
```

W fazie wykonywania BEGIN zaznacza początek sekwencji, której wykonanie ma być powtarzane. Służy jako punkt powrotu z odpowiedniego UNTIL, AGAIN lub REPEAT. Przy wykonywaniu UNTIL powrót do BEGIN następuje wtedy, gdy tuż przed UNTIL wartość na stosie równa jest zeru (znacz-

nik logiczny ff) . W pozostałych dwóch konstrukcjach powrót następuje zawsze czyli zatrzymywanie pętli osiąga się innymi środkami. W fazie kompilacji BEGIN zostawia swój. adres powrotu, a n służy do kontroli błędów.

BL --- c
Stała zostawiająca na stosie wartość spacji w kodzie ASCII (32 dec, 20 hex) .

BLANKS adr n ---
Wypełnia spacjami obszar pamięci od adr na długości n bajtów.

BLK --- adr
Zmienna użytkownika. Podaje adres, pod którym znajduje się numer bloku bieżąco interpretowanego. Gdy zmienna ma wartość 0, wprowadzanie następuje z buforu terminalu.

BLOCK n --- adr
Zostawia na stosie adres w pamięci buforu zawierającego blok numer n. Jeżeli blok nie znajduje się jeszcze w pamięci, BLOCK wywołuje jego wprowadzenie z dyskietki do jednego z buforów. Jeżeli plik, który zajmował ten bufor został oznaczony, jako zmieniony z pomocą UPDATE, jest on zapisywany na dyskietkę przed wprowadzeniem bloku nr n. Patrz także R/W, UPDATE, FLUSH.

BOOT ---
Wykonywane działa tak samo, jak włączenie komputera.

BRANCH --- adres względny
Procedura systemowa stosowana przy kompilacji odgałęzień, które wykonuje, gdy napotka wartość niezerową (tf). Wykonuje przejście pod adres względny, tzn. obliczony w stosunku do adresu BRANCH.

BUFFER n --- adr
Zostawia na stosie adres buforu skojarzonego z (być może, nowym) blokiem o numerze n.

C! b adr ---
Wpisuje 8-bitów b pod adresem adr.

C, b ---
Wpisuje 8-bitowe b do najbliższego dostępnego bajtu słownika zwiększając jego wskaźnik o 1,

C? adr ---
Drukuje wartość bajtu spod adr.

C@ adr --- b
Zostawia na stosie 8-bitową zawartość adresu.

C/L --- n
Stała zostawiająca na stosie długość linii w bajtach. Używana przez edytor.

CFA pfa --- cfa
Przekształca adres pola parametrów definicji w adres jej pola kodu czyli odejmuje 2.

CLIT --- a
definicji dwukropkowej CLIT jest automatycznie kompilowane przed każdą wprowadzaną do niej liczbą 80-bitową. Późniejsze wykonanie CLIT powoduje, że zawartość następnego bajtu jest przenoszona na stos.

CMOVE adr1 adr2 n ---
Przesuwa n bajtów poczynając od adr1 do odcinka pamięci zaczynającego się. od adr2. Zawartość adr1 przesuwana jest jako pierwsza. Gdy obszar przeznaczenia znajduje się wyżej i obszary częściowo się pokrywają, część informacji może być utracona.

CMOVE> adr1 adr2 n ---
Działa jak CMOVE, jednak przesuwanie bajtów zaczyna od najwyższego adresu odcinka, co przy przesuwaniu w górę na obszar częściowo pokrywający się zapobiega utracie informacji. Stosowane w niektórych systemach.

CODE ---
Tworzy wejście słownikowe definiowane w całości w kodzie maszynowym.

COLD ---
Procedura zimnego startu przesuwająca wskaźnik słownika DP poniżej nowo zdefiniowanych słów i powodująca ponowny start za pośrednictwem ABORT. COLD może być wywołane w celu usunięcia programów użytkownika i oczyszczenia buforów oraz ponownego startu.

COMPILE ---
Gdy słowo zawierające COMPILE jest wykonywane, adres wykonania słowa następującego po COMPILE jest. kopiowany (kompilowany) do słownika, a wskaźnik słownika jest odpowiednio zwiększany. Umożliwia to stosowanie specjalnych metod kompilacji.

CONSTANT n ---
Słowo definiujące używane w następującej postaci:

```
n CONSTANT cccc
```

Tworzy słowo o nazwie cccc, którego pole parametrów zawiera n. Gdy cccc jest następnie wykonywane, wartość n wprowadzana jest na stos.

CONTEXT --- adr
Zmienna użytkownika. Zostawia na stosie adres zawierający wskaźnik podsłownika, w którym poszukiwanie słów podejmowane jest w pierwszej kolejności.

CONVERT d1 adr1 --- d2 adr2
Przekształca stosownie do aktualnej podstawy układu liczbowego ciąg znaków numerycznych zaczynający się pod adr1 w podwójną liczbę binarną gromadząc wynik w drugiej liczbie na stosie. Konwersja kończy się, gdy CONVERT napotka znak, który nie jest cyfrą. Jego adres podawany jest - w adr2.

COPY n1 n2 ---
Kopiuje zawartość ekranu n1 na ekran n2.

COUNT adr1 --- adr2 n
Umieszcza na stosie adres pierwszego bajtu łańcucha znaków oraz podaną w nim długość tego łańcucha w bajtach. Zakłada się przy tym, że pierwszy bajt pod adr1 zawiera n i że tekst zaczyna się od następnego bajtu. Zazwyczaj COUNT przygotowuje dane dla TYPE.

CR ---
Przenosi druk do następnej linii.

CREATE Słowo definiujące stosowane w następującej postaci:

CREATE cccc Tworzy w połączeniu z takimi słowami, jak CODE i CONSTANT, nowe wejście do słownika o nazwie cccc. Nowe słowo tworzone jest w podsłowniku CURRENT.

CSAVE ---
Powoduje zapisanie całego znajdującego się. W pamięci programu sterującego i słownika na kasetę.

CSP --- adr
Zostawia na stosie adres zmiennej czasowo przechowującej wskaźnik stosu.

CURRENT --- adr
Zestawia na stosie adres zmiennej wskazującej podsłownik do nowych definicji.

D! d adr ---
Wpisuje 32-bitowe d pod adresem adr.

D+ d1 d2 --- d3
Zostawia na stosie d3 stanowiące sumę d1 i d2.

D+- d1 n --- d2
Nadaje liczbie podwójnej d1 znak liczby n i zostawia ją jako d2.

D. d ---
Drukuje liczbę podwójną ze znakiem podaną stosownie do aktualnej BASE, tzn., na przykład, jako liczbę w układzie dziesiątkowym, szesnastkowym lub innym, jaki w danej chwili stosujemy.

D.R d n ---
Drukuje liczbę podwójną ze znakiem w polu o szerokości n dosuniętą do prawej krawędzi pola rozpoczętego na aktualnej pozycji kursora.

D< d1 d2 --- f
Zostawia na stosie znacznik logiczny tf, gdy d1 jest mniejsze, niż, d2 lub ff w przeciwnym wypadku.

D@ adr --- d
Zastępuje na stosie adres znajdującą się pod nim liczbą podwójną d.

DABS d --- ud
Zostawia wartość bezwzględną ud liczby podwójnej. d.

DECIMAL Ustala podstawę konwersji dla wprowadzania i wyprowadzania na 10 (układ dziesiętny) .

DEFINITIONS ---
Używane w postaci:

cccc DEFINITIONS

Przekształca podsłownik ccccc CURRENT w CONTEXT. Pozwala wprowadzać do cccc nowe definicje, a nawet całe dalsze podsłowniki.

DIGIT n1 n2 --- n3 tf
n1 n2 --- ff
Przekształca dolny bajt n1 rozumiany jako kod ASCII stosownie do podstawy podanej w n2 zostawiając binarny równoważnik tej liczby n3 oraz znacznik tf. Jeżeli konwersja nie jest możliwa, zostawia ff.

DLITERAL d --- d wykonywanie
d --- kompilacja
Podczas kompilacji wprowadza do słownika poprzedzającą liczbę podwójną. Późniejsze wykonanie spowoduje umieszczenie tej liczby na stosie. W bezpośrednim wykonaniu jest operacją pustą, bez jakichkolwiek efektów.

DMAX d1 d2 --- d3
Zostawia na stosie większą z dwóch liczb podwójnych.

DMIN d1 d2 --- d3
Zostawia na stosie mniejszą z dwóch liczb podwójnych.

DMINUS d --- -d
Przekształca liczbę podwójną w jej uzupełnienie do dwóch.

DNEGATE To samo co DMINUS.

DO n1 n2 --- wykonywanie
adr n --- kompilacja
W definicji dwukropkowej rozpoczyna pętlę liczoną:

DO ... LOOP

DO ... n3 +LOOP

W czasie wykonywania definicji D otwiera sekwencję, której powtarzające się wykonanie kontrolowane jest przez granicę pętli n1 i jej wskaźnik o początkowej wartości n2. DO przenosi obie te wartości ze stosu głównego na stos powrotów. Gdy wykonanie definicji dociera do LOOP wskaźnik jest zwiększany o 1, a gdy do +LOOP - o n3. Jeżeli nowy wskaźnik jest nadal mniejszy od granicy pętli, wykonanie powraca tuż są DO; w przeciwnym wypadku działanie pętli kończy się, a jej parametry są kasowane i wykonanie przechodzi do następnej części definicji

lub poza jej obręb. Zarówno n1, jak i n2 określane są w fazie wykonywania 1 mogą być wynikiem innych operacji. W obrębie pętli słowo "I" może być wykorzystane do kopiowania bieżącej wartości wskaźnika n2 na stos. Patrz także: I, J, LOOP, +LOOP, LEAVE.

DOES> ---
Określa działanie w fazie wykonywania w obrębie słowa zdefiniowanego na wyższym poziomie. Zmienia pole kodu i pierwszy parametr nowego słowa, by wykonać sekwencję adresów skompilowanego słowa znajdującą się za DOES> . Używane jest w powiązaniu z <BUILDS . Gdy część za DOES> jest wykonywana, zaczyna się od adresu na stosie pierwszego parametru nowego słowa. Umożliwia to interpretację przy użyciu tego obszaru lub jego treści. Typowe zastosowania obejmują asembler FORTH-a, wielowymiarowe tablice i tworzenie kompilatora.

DP --- adr
Zmienna użytkownika. Wskaźnik słownika zawierający adres najbliższej wolnej komórki pamięci nad słownikiem. Wartość tę podaje bezpośrednio HERE, a zmienia ALLOT.

DPL --- adr
Zmienna użytkownika. Zostawia na stosie adres zmiennej zawierającej liczbę cyfr, które we wprowadzanej podwójnej liczbie całkowitej znajdują się na prawo od kropki dziesiętnej. DPL może być także stosowane do określania miejsca kropki dziesiętnej w wyprowadzanych liczbach sformatowanych przez użytkownika. Domyślna wartość DPL przy wprowadzaniu liczb pojedynczych wynosi -1.

DROP n ---
Usuwa liczbę ze stosu.

DUMP adr n ---
Drukuje zawartość n bajtów pamięci poczynając od adr. Wszystkie dane drukowane są w aktualnym układzie liczbowym określonym w BASE.

DUP n --- n n
Kopiuje liczbę znajdującą się na szczycie stosu.

EDITOR ---
Czyni podsłownik edytora podsłownikiem CONTEXT.

ELSE ---
Występuje w definicji dwukropkowej w postaci:

IF ... ELSE ... ENDIF

W fazie wykonywania ELSE wykonuje następującą po nim część definicji wtedy, gdy wartość na szczycie stosu przed IF wynosiła zero i kończy działanie po dojściu do ENDIF. Nie wywiera wpływu na stos.

EMIT c ---
Przekazuje znak odpowiadający c w kodzie ASCII na wskazane urządzenie, np. drukuje ten znak na ekranie. Jako c wykorzystuje dolny bajt liczby na stosie.

EMPTY ---
Kasuje wszystkie wejścia słownikowe aż do FENCE.

EMPTY-BUFFERS ---
Zeruje wszystkie bufory bloków i oznacza je jako wolne. Nie zapisuje na dyskietkę zmienionych bloków.

END To samo co UNTIL. Stosowane w FORTH-83.

ENDIF adr n --- kompilowanie
Występuje w definicji dwukropkowej w postaci:

IF ... ENDIF

IF ... ELSE ... ENDIF

W fazie wykonywania ENDIF służy jedynie jako punkt docelowy odgałęzienia do przodu od IF lub. ELSE. Równoważnikiem ENDIF jest THEN.

ERRASE adr n ---
Wypełnia zerami n bajtów poczynając od adr.

ERROR n ---
Drukuje komunikat o błędzie numer n i wykonuje ponowny gorący start. Fig-FORTH zachowuje przy tym zawartości IN i BLK, by umożliwić ustalenie miejsca błędu. Końcowym działaniem jest wykonanie QUIT.

EXECUTE adr ---
Wykonuje definicję, której pole kodu znajduje się na stosie. Adres pola kodu nazywany jest także adresem kompilacji.

EXIT ---
Podczas wykonywania słowa określonego z pomocą definicji dwukropkowej kończy jego wykonanie.

EXPECT adr n ---
Przenosi znaki z terminalu pod adres, dopóki nie zostanie wykonany Return bądź nie zostanie przesłanych n znaków. Na końcu tekstu dodaje jedno lub więcej zer.

FENCE --- adr
Zmienna użytkownika. Zawiera adres, poniżej którego nie działa komenda FORGET. Aby skasować fragment słownika poniżej tego punktu, użytkownik musi zmienić zawartość FENCE.

FILL adr n b ---
Wypełnia n bajtów pamięci poczynając od adr wartością wskazaną w b.

FIRST --- n
Stała, która zostawia na stosie adres pierwszego (najniższego) buforu bloków.

FLUSH ---
Zapisuje na dyskietkę zawartość wszystkich buforów zaznaczonych jako zmienione z pomocą UPDATE.

Edytor na ogół automatycznie zaznacza zmienione bloki. FLUSH należy wykonać przed wyjściem z FORTH-a, by zapobiec utracie dokonanych zmian. Równoważnikiem FLUSH jest SAVE-BUFFERS.

FORGET ---

Wykonywane w postaci:

FORGET cccc

Usuwa ze słownika definicję o nazwie cccc wraz z wszystkimi wejściami słownikowymi wprowadzonymi później niż cccc.

FORTH ---

Nazwa pierwotnego, głównego podsłownika FORTH-a. Stanowi zarazem słowo, którego wykonanie powoduje, że podsłownik główny staje się podsłownikiem CONTEXT, tzn. poszukiwania zaczynają się od niego. Jeżeli nie zostaną zdefiniowane odrębne podsłowniki użytkownika, wszystkie jego nowe definicje staną się częścią podsłownika FORTH. Jest on natychmiastowy, to znaczy w przypadku tworzenia nowych definicji dwukropkowych system korzysta z niego w fazie kompilacji.

HERE --- adr

Zostawia na stosie adres pierwszej wolnej komórki nad słownikiem.

H. n ---

Wyprowadza liczbę w układzie szesnastkowym i powraca do aktualnej podstawy konwersji.

HEX ---

Zmienia podstawę liczbową konwersji na 16.

HLD --- adr .

Podaje adres zmiennej, w której przechowywany jest adres ostatniego znaku tekstowego wykorzystywanego przy konwersji podczas formatowanego wyprowadzania liczb w postaci ciągu znaków.

Po uruchomieniu konwersji z pomocą słowa <# do HLD kopiowany jest z PAD adres buforu wyprowadzania tekstu. Po każdym wyprowadzonym znaku cyfry zawartość HLD jest zmniejszana o 1.

HOLD c ---
Używane między <# a #> w celu "wsunięcia" znaku w kodzie ASCII do wyprowadzanych łańcuchów cyfr. Np. 46 HOLD umieści w wybranym miejscu kropkę dziesiętną, a 44 HOLD - przecinek.

I --- n
Używane w pętlach liczonych DO ... LOOP i DO ... +LOOP kopiuje na stos wskaźnik pętli.

ID lub ID. adr ---
Drukuje nazwę definicji spod podanego adresu jej pola nazwy (nfa) .

IF f --- wykonywanie
--- adr n kompilacja
Występuje w definicji dwukropkowej w formie=

IF ... ENDIF

IF ... ELSE ... ENDIF

Podczas wykonywania, gdy IF napotka na stosie wartość niezerową, wykonywana jest część definicji bezpośrednio za IF. Gdy napotka 0, wykonywana jest część za ELSE lub w przypadku jego niezastosowania - za ENDIF. W fazie kompilowania IF kompiluje OBRANCH i rezerwuje przestrzeń na przesunięcie pod adr.

IMMEDIATE ---
Powoduje, że ostatnio stworzona definicja, gdy zostanie napotkana podczas kompilacji, będzie natychmiast wykonana, a nie kompilowana. Do tej kategorii definicji należą m. in. IF i DO. Użytkownik może wymusić skompilowanie definicji natychmiastowej (immediate) poprzedzając zawarte w niej słowa słowem [COMPILE].

IN --- adr
Zmienna użytkownika. Zostawia na stosie adres bieżącej pozycji w buforze wprowadzania z terminalu lub stacji dyskietek. WORD wykorzystuje i zmienia wartość IN.

INDEX n1 n2 ---
Drukuje zerowe linie z ekranów o numerach od n1 do n2. Zwyczajowo są to linie komentarzy-nagłówków określających zawartość ekranu.

INTERPRET ---
Rozpoczyna interpretację bloku, którego numer zawiera BLK, poczynając od znaku wskazanego w IN.

IP ---
Wskaźnik instrukcji interpretera. Podaje adres słowa, które będzie używane jako następne. Wykorzystywany w asemblerze.

J --- n
Wskaźnik pętli wewnętrznej przy zagnieżdżeniach.

KEY --- c
Zostawia na stosie wartość ASCII znaku po naciśnięciu wywołującego go klawisza. Czeka na naciśnięcie klawisza.

LATEST --- adr
Zostawia na stosie adres pola nazwy słowa znajdującego się na szczycie podsłownika CURRENT

LEAVE ---
Służy do wcześniejszego zakończenia pętli DO ... LOOP i DO...+LOOP. Nadaje granicy pętli wartość jej wskaźnika. Wskaźnik nie zostaje zmieniony, toteż pętla zostanie zakończona dopiero po napotkaniu LOOP lub +LOOP.

LFA pfa --- lfa
Przekształca adres pola parametrów definicji w słowniku w adres jej pola łącznika. Innymi słowy, zmniejsza adres pola parametrów o 4.

LIMIT --- n

Zostawia na stosie adres komórki położonej bezpośrednio nad najwyższą komórką dostępną w buforze stacji dyskietek.

LIST n ---

Wyświetla tekst ekranu numer n z wybranego urządzenia zewnętrznego. W czasie tego działania i po jego zakończeniu zmienna SCS zachowuje wartość n.

LIT --- n

W definicji dwukropkowej LIT jest automatycznie kompilowane przed każdą liczbą 16-bitową zawartą we wprowadzanym tekście definicji. Późniejsze wykonanie LIT powoduje, że następująca po nim liczba wprowadzana jest na stos. W fig-FORTH LIT jest pierwszym słowem w słowniku.

LITERAL n --- n wykonywanie

n --- kompilacja

Podczas kompilacji kompiluje LIT, a następnie liczbę n. Późniejsze wykonanie spowoduje umieszczenie tej liczby na stosie. Jest w definicjach słowem natychmiastowym. W bezpośrednim wykonaniu jest operacją pustą.

LOAD n ---

Rozpoczyna interpretację ekranu numer n. Zakończenie ładowania następuje wraz z końcem ekranu lub napotkaniem słowa ;S.

LOOP adr n --- kompilacja

Używane w definicji dwukropkowej w postaci:

DO ... LOOP

W fazie wykonywania pętli kontroluje powrót do DO na podstawie wskaźnika pętli i jej granicy. Po każdym wykonaniu wskaźnik pętli zwiększany jest o 1 i porównywany z granicą. Gdy wskaźnik zrówna się z granicą lub ją przekroczy, pętla zakończy działanie, a jej parametry będą skasowane.

Podczas kompilacji LOOP kompiluje (LOOP) oraz wykorzystuje adres do obliczenia wielkości przejścia do DO. Wówczas n służy do kontroli błędów.

M* n1 n2 --- d
Mnoży dwie liczby pojedynczej długości i zostawia na stosie liczbę podwójną stanowiącą ich iloczyn.

M/ d n1 --- n2 n3
Zostawia na stosie resztę n2 ze znakiem d oraz iloraz n3 z dzielenia całkowitoliczbowego dzielnej d przez dzielnik n1.

M/MOD ud1 u2 --- u3 ud4
Operacja nad liczbami bez znaku podwójnej i pojedynczej dokładności. Zostawia na stosie iloraz podwójnej długości bez znaku ud4 i pojedynczej długości resztę bez znaku u3 z dzielenia podwójnej długości dzielnej bez znaku ud1 przez pojedynczej długości dzielnik u2.

MAX n1 n2 --- n3
Zostawia na stosie większą z dwóch liczb.

MESSAGE n ---
Drukuje na wybranym urządzeniu tekst linii n z ekranu nr 4 w stacji dyskietek. Ekrany 4 i 5 według standardu (nie zawsze stosowanego) zawierają komunikaty o błędach, n może być dodatnie lub ujemne. MESSAGE można używać do drukowania powtarzających się tekstów, jak nagłówki stron lub pism. Jeżeli WARNING ma wartość 0, (stacja dyskietek niedostępna), MESSAGE wydrukuje tylko liczbę.

MIN n1 n2 --- n3
Zostawia na stosie mniejszą z dwóch liczb.

MINUS n1 --- n2
Zostawia na stosie uzupełnienie liczby do dwóch.

MOD n1 n2 --- n3
Zostawia resztę z dzielenia n1 przez n2 ze znakiem n1.

MOVE adr1 adr2 n ---
Przemieszcza zawartość n 16-bitowych jednostek pamięci zaczynających się pod adr1 do n jednostek zaczynających się pod adre2. Zawartość adr1 przesuwana jest jako pierwsza.

N d,n lub b ---
Stosowana w niektórych implementacjach 8-bajtowa strefa pamięci służąca do przechowywania danych użytkownika.

NEGATE To samo co MINUS.

NFA pfa --- nf a
Przekształca adres pola parametrów definicji w adres jej pola nazwy.

NOOP ---
"No operation" FORTH-a. Operacja pusta.

NOT n1 --- n2
Zostawia na stosie uzupełnienie n1 do 1 (inwersja bitów).

NUMBER adr --- d
Przekształca łańcuch znaków spod adr poprzedzonych bajtem długości łańcucha w liczbę podwójną ze znakiem stosując bieżący układ liczbowy. Jeżeli w tekście napotka kropkę, jej pozycja zostanie podana w DPL bez innych następstw. Gdy konwersja liczbowa nie jest możliwa, wydawany jest komunikat o błędzie.

OFFSET ---
Zmienna wykorzystywane przez system przy obliczaniu skoku pętli.

O. n ---
Wyprowadza liczbę w układzie ósemkowym i powraca do aktualnej bazy konwersji.

OR n1 n2 --- n3
Zostawia na stosie wynik wykonania logicznego OR na bitach liczb n1 i n2.

OTHERWISE To samo co ELSE.

OUT --- adr
Zmienna użytkownika. Zawiera wartość zwiększaną przez EMIT. Użytkownik może zmieniać i sprawdzać OUT w celu kontrolowania formatu wyświetlania.

OVER n1 n2 -- n1 n2 n1
Kopiuje drugą wartość stosu i umieszcza kopię na jego szczycie.

PAD --- adr
Podaje adres szczytu buforu wyprowadzania tekstu położonego w ustalonej odległości od HERE, szczytu słownika. PAD ma zwykle 84 bajty długości

PFA nfa --- pfa
Przekształca adres pola nazwy skompilowanej definicji w adres jej pola parametrów.

PICK n1 --- n2
Kopiuje wartość z pozycji n1 +1 od szczytu na szczyt stosu.

PREV --- adr
Zostawia na stosie adres zmiennej zawierającej adres buforu bloku, który. był lub jest wykorzystywany jako ostatni.

QUERY ---
Wprowadza 80 znaków tekstu lub do Return z terminalu operatora do buforu terminalu. Tekst umieszczany jest pod adresem zawartym w TIB, a IN otrzymuje wartość 0.

QUIT ---
Zeruje stos powrotów, zatrzymuje kompilację i zwraca kontrolę do terminalu operatora. Nie podaje żadnej informacji ani nagłówka.

R --- n
Umieszcza na szczycie stosu głównego kopię wartości ze szczytu stosu powrotów nie kasując jej.

R# --- adr
Zmienna użytkownika. Podaje pozycje kursora w buforze ekranu. Stosowana w edytorze.

R/W adr blk f ---
Standardowa w fig-FORTH procedura sterowania zapisem i odczytem z dyskietki: adr określa adres wskazanego bufora źródłowego lub docelowego , blk jest numerem bloku; f=0 powoduje zapis, a f=1 czytanie z dyskietki.

R> --- n
Usuwa górną wartość ze stosu powrotów i umieszcza ją na stosie głównym. Patrz także >R i R.

RO --- adr
Zmienna użytkownika. Zawiera początkowy adres stosu powrotów. Patrz RP!

R@ To samo co R.

RECURSE ---
Kompiluje adres pola kodu słowa, tak iż może być ono wykonywane rekursyjnie.

REMEMBER ---
Powoduje, że następne wprowadzone słowo, gdy będzie wykonywane, skasuje siebie i wszystkie po nim zdefiniowane.

REPEAT adr n --- kompilacja
Używane w definicji dwukropkowej w postaci :

BEGEN ... WHILE ... REPEAT

W fazie wykonywania REPEAT powoduje bezwarunkowe przejście wstecz do odpowiedniego BEGIN. Podczas kompilacji REPEAT kompiluje BRANCH i przesunięcie od HERE do adresu adr; n służy do kontroli błędów.

ROLL n1 n2 ... nn n --- n2 ... nn n1
Wykonuje rotację polegającą na przesunięciu wartości znajdującej się na miejscu n1+1 od szczytu stosu na jego szczyt i obniżeniu powstałych wartości. Tak wiec ROT - to 2 ROLL, a SWAP - to 1 ROLL.

ROT n1 n2 n3 --- n2 n3 n1
Wykonuje rotację trzech pierwszych liczb umieszczając trzecią na szczycie stosu.

RP! ---
Inicjalizuje stos powrotów.

RP@ --- adr
Zostawia na stosie bieżący stan wskaźnika stosu powrotów.

S->D n --- d
Przekształca z zachowaniem znaku liczbę pojedynczej długości w podwójną.

SO --- adr
Zmienna użytkownika. Zawiera początkową wartość wskaźnika stosu. Patrz SP!

SAVE ---
Powoduje, że interpretator FORTH-a wraz z całym znajdującym się w danej chwili w pamięci słownikiem zostaje zapisany na dyskietkę poczynając od sektora 1 w postaci ładującej się automatycznie. Posiadając komendę można tworzyć dowolnie rozszerzane (lub ograniczane) wersjo słownika.

SAVE-BUFFERS to samo co FLUSH.

SCR --- adr
Zmienna użytkownika. Zawiera numer ekranu ostatnio wywołanego z pomocą LIST.

SET n adr ---
Definiuje następne wprowadzone słowo jako komendę, która wpisze n pod adr.

SIGN n d --- d
Wprowadza znak minusa przed liczbą poddaną konwersji i wyprowadzaną do buforu wyprowadzania tekstu. Jeżeli n jest ujemne, n będzie skasowane, natomiast zachowana zostanie na stosie liczba d. Musi być stosowane między <# a #>.

SMUDGE ---
Wprowadza do pola nazwy tworzonej definicji "bit znamienia". Sprawia on, że nie zakończone definicje są pomijane podczas przeszukiwania słownika.

SP! ---
Inicjalizuje wskaźnik stosu.

SP@ --- adr
Zostawia na stosie adres najwyższej pozycji na stosie tuż przed wykonaniem SP@.

SPACE ---
Wyprowadza spację ASCII.

SPACES n---
Wyprowadza n spacji ASCII.

STATE ---
Zmienna użytkownika. Określa stan kompilacji. Wartość niezerowa wskazuje na kompilację. Sama wartość może zależeć od implementacji.

SWAP n1 n2 --- n2 n1
Zamienia miejscami dwie górne wartości na stosie.

TEXT b ---
Wprowadzany tekst aż do ogranicznika b przesuwa pod PAD poprzedzając bajtem długości tekstu.

THEN To samo co ENDIF.

THRU n1 n2 ---
Ładuje kolejne bloki od numeru n1 do n2.

TIB --- adr
Zmienna użytkownika. Zawiera adres buforu terminalu.

TOGGLE adr b ---
uzupełnia zawartość adresu o wzór bitowy b.

TRAVERSE adr1 n --- adr2
Ustala adres przeciwległego pola nazwy definicji, adr1 jest adresem bajtu długości nazwy albo jej ostatniej litery. Gdy n=1, ruch odbywa się w stronę wyższych adresów, a gdy n=-1 w stronę niższych. W efekcie TRAVERSE pozostawia na stosie adres końca lub początku pola nazwy.

TRIAD scr ---
Drukuje na wybranym urządzeniu trzy ekrany zawierające ekran o numerze scr. Zaczyna od ekranu, którego numer podzielny jest przez 3. Wyprowadzany tekst przygotowany jest do wykorzystania w źródłowych rekordach czyli zbiorach tekstowych i obejmuje również tekst dolnej paginy, który pobiera z linii 15 ekranu lub innego wykorzystywanego przez MESSAGE. Patrz również MESSAGE.

TYPE adr n ---
Drukuje na wskazanym urządzeniu n bajtów pamięci od adr.

U* u1 u2 --- ud
Zostawia na stosie liczbę podwójnej długości bez znaku ud stanowiącą iloczyn dwóch liczb pojedynczej długości bez znaku.

U/ ud u1 --- u2 u3
Zostawia na stosie wynik operacji na liczbach bez znaku: resztę u2 i iloraz u3 z dzielenia liczby podwójnej ud przez u1.

U< u1 u2 --- f
Zostawia na stosie znacznik logiczny porównania dwóch liczb bez znaku. Gdy u1 jest mniejsze od u2 zostawia tf, w przeciwnym wypadku ff. u1 i u2 są 16-bitowymi binarnymi liczbami całkowitymi bez znaku. U< musi być stosowane, gdy

porównuje się adresy lub liczby, które mogą być większe niż 32767.

U. u ---
Drukuje 16-bitową liczbę bez znaku u w bieżącym układzie liczbowym określonym w BASE. Używane zamiast kropki, gdy liczba może przekroczyć 32767 czyli 7fff hex.

U? adr --- u
Drukuje 16-bitową liczbę bez znaku u spod adr.

UNTIL f --- wykonywanie
adr n --- kompilacja
Używane w definicjach dwukropkowych w formie:

BEGIN ... UNTIL

W fazie wykonywania UNTIL kontroluje warunkowe odgałęzianie wstecz do odpowiedniego BEGIN. Gdy f wskazuje "fałsz", wykonanie powraca do miejsca tuż za BEGIN. W przeciwnym wypadku wykonanie przechodzi poza pętlę, tuż za UNTIL. W fazie kompilacji UNTIL kompiluje OBRANCH z przesunięciem od HERE do adr. Wartość n służy do kontroli błędów.

UP --- adr
Zmienna użytkownika. Zawiera adres początku obszaru pamięci, w którym umieszczane są zmienne użytkownika. Patrz USER.

UPDATE ---
Wprowadza oznaczenie wskazujące, że blok ostatnio wykorzystywany, podany w PREV, został przeredagowany. Zapobiega to utraceniu bloku. W chwili, gdy bufor będzie potrzebny do wprowadzenia następnego bloku, blok zaznaczony przez UPDATE zostanie automatycznie przeniesiony na dyskietkę.

USE --- adr
Zmienna zawierająca adres buforu bloku, który

będzie użyty jako następny po ostatnio zapisanym.

USER n ---
Słowo definiujące używane w postaci:

```
n USER cccc
```

Tworzy zmienną użytkownika cccc. Pole parametrów cccc zawiera n jako stałe przesunięcie (offset) dla tej zmiennej użytkownika w stosunku do UP. Gdy cccc jest później wykonywane, umieszcza na stosie sumę tego przesunięcia i adresu początku obszaru zmiennych użytkownika, tworząc efektywny adres, pod którym użytkownik może umieszczać wartości swojej zmiennej.

VARIABLE ---
Słowo definiujące używane w postaci:

```
n VARIABLE cccc
```

Gdy VARIABLE jest wykonywane, tworzy definicję cccc z jej polem parametrów inicjalizowanym wartością n. Gdy cccc jest później wykonywane, adres jej pola parametrów, zawierającego n, umieszczany jest na stosie, co zapewnia dostęp do tej zmiennej zarówno dla jej pobierania, jak i zmiany jej wartości. Pamiętać należy, że pisząc cccc wywołujemy adres zmiennej, a nie jej wartość.

VLIST ---
Listuje nazwy definicji w podsłowniku CONTEXT. Nazwy definicji wyświetlane są poczynając od najpóźniej wprowadzonych. Na końcu zawsze wyświetlany jest podsłownik główny FORTH. Przykład: aby obejrzeć nazwy definicji podsłownika EDITOR, gdy jest wprowadzony, należy napisać EDITOR VLIST.

VOC-LINK --- adr
Zostawia na stosie adres zmiennej zawierającej początek drzewa łączącego wszystkie podsłowniki.

VOCABULARY ---

Słowo definiujące używane w postaci:

VOCABULARY cccc

Tworzy podsłownik o nazwie cccc. Gdy napiszemy później cccc, podsłownik ten stanie się podsłownikiem CONTEXT, to znaczy przeszukiwanym jako pierwszy przez INTERPRET. Sekwencja cccc DEFINITIONS sprawi, że stanie się on także podsłownikiem CURRENT, a więc będzie przygotowany do wprowadzenia nowych definicji. W fig-FORTH cccc zostaje sprzężone z wszystkimi definicjami podsłownika, w którym samo zostało zdefiniowane. Wszystkie podsłowniki są ostatecznie sprzężone z głównym podsłownikiem FORTH. W niektórych konwencjach nazwy podsłowników należy deklarować jako natychmiastowe, np.

VOCABULARY EDITOR IMMEDIATE

W ---

Adres aktualnie wykonywanego słowa. Używane w asemblerze.

WARNING --- adr

Zmienna użytkownika. Kontroluje możliwość posłużenia się komunikatami drukowanymi z pomocą MESSAGE. Gdy równa jest 4, stacja dyskietek jest czynna i ekran nr 4 (w niektórych implementacjach inny) w stacji nr 0 stanowi miejsce do umieszczania komunikatów. Gdy WARNING = 0, stacja nie jest czynna i komunikaty przybierają postać reprezentujących je liczb. Gdy wartość wynosi -i, wykonuje się ABORT dla procedur użytkownika. Patrz także MESSAGE i ERROR.

WHILE f --- wykonanie

adr1 n1 - adr1 n1 adr2 n2

Występuje w definicjach dwukropkowych w postaci:

BEGIN ...WHILE ... REPEAT

W czasie wykonania WHILE dokonuje warunkowego rozgałęzienia zależnie od znacznika logicznego f.

Jeżeli ma on wartość niezerową, WHILE powoduje kontynuowanie części za nim aż do REPEAT, po czym następuje powrót tuż za BEGIN. Jeżeli znacznik wskazuje "fałsz", wykonanie opuszcza pętlę przechodząc tuż za REPEAT.

WIDTH --- adr
Zmienna użytkownika. Określa maksymalną liczbę znaków nazwy definicji. Wartość domyślna wynosi zazwyczaj 31. Użytkownik może zmniejszyć z pomocą WIDTH liczbę znaków nazw.

WORD c ---
Czyta znaki z interpretowanego strumienia wprowadzania, dopóki nie napotka ogranicznika c, i umieszcza łańcuch tych znaków poczynając od buforu słownika pod HERE. WORD lokuje w pierwszym bajcie liczbę znaków, następnie łańcuch znaków, a na końcu dwie lub więcej spacji. Gdy BLK ma wartość 0, wprowadzanie dokonywane jest z buforu terminalu. Gdy BLK ma wartość niezerową, tekst pobiera się z bloku dyskietki o numerze podanym w BLK. Patrz także BLK i IN.

XOR n1 n2 --- XOR
Zostawia wynik obliczenia na bitach n1 i n2 relacji nierównoważności (exclusive or) .

XSAVE ---
Zmienna do przechowywania wartości indeksu X. Stosowana w asemblerze.

[---
Używane w definicji dwukropkowaj w postaci:

: cccc słowa dalsze-słowa ;

Zawiesza kompilację. Słowa po[są natychmiast wykonywane, a nie kompilowane. Umożliwia to obliczenia lub kompilowanie sytuacji wyjątkowych przed wznowieniem kompilacji z pomocą] . Patrz także LITERAL,] .

[COMPILE] ---

Używane w definicji dwukropkowej w postaci:

```
: cccc COMPILE FORTH ;
```

Wymusza kompilację słowa natychmiastowego, które w przeciwnym wypadku byłoby wykonywane podczas kompilacji. W podanym wyżej przykładzie słownik FORTH wybrany będzie przy wykonywaniu cccc, a nie w fazie kompilacji.

] ---

Wznawia kompilację w celu dokończenia definicji dwukropkowej.

SPIS TREŚCI

Przedmowa 3
Rozdział 1. DLACZEGO FORTH..... 6
1.1 Trochę historii 5
1.2 Jaki jest FORTH? 6
1.3. Podstawowe cechy 8
1.3.1 Szybkość 8
1.3.2 Półkompilacja 10
1.3.3 Pamięcioooszczędność 11
1.3.4 Rozszerzalność..... 12
1.3.5 Dialogowość..... 13
1.3.6 Inne cechy 13
1.4 Gama zastosowań 14
1.6 Implementacje 16
1.6 Oprogramowanie systemowe 17
CZĘŚĆ I PODSTAWY..... 19
Rozdział 2 PIERWSZE KROKI 21
2.1 Laik przy komputerze..... 21
2.2 Poznajmy pętlę 24
2.3 Podstawowe reguły dialogu 24
Rozdział 3 W ŚWIECIE LICZB 27
3.1 Rodzaje liczb 27
3.2 Potęga stosów 29
3.3 Odwrotna notacja polska /ONP/ 31
3.4 Manipulowanie liczbami na stosie..... 33
3.4.1 Budowa stosu 33
3.4.2 Wprowadzanie 1 wyprowadzanie liczb 33
3.4.3 Przestawienia 35
3.4.4 Jak kontrolować stos?..... 37
3.5 Stos powrotów..... 39
3.6 Arytmetyka udoskonalona..... 41
3.6.1 Liczby pojedynczej długości 41

3.6.2 Liczby podwójnej długości 43
3.7 Jak zmienić układ liczbowy? 44
3.8 Liczby na stosie i w pamięci 47
Rozdział 4 SŁOWA 51
4.1. Definiowanie 51
4.2 Wprowadzanie i odczytywanie wartości zmiennej... 52
4.3 i gdzie to jest? 54
4.4 Budowa słowa 55
4.5 Dostęp do słowa 62
Rozdział 5 KONSTRUKCJE PROGRAMOWANIA 64
5.1 Czym jest programowanie strukturalne? 64
5.2 Porównania 65
5.2.1 Porównywanie liczb pojedynczej długości 66
5.2.2 Porównywanie liczb podwójnej długości 67
5.3 Jeżeli... to 68
5.4 Pętla podstawowa..... 70
5.5 Pętla liczone: DO...LOOP i DO...+LOOP 72
5.6 Przykład: tabliczka mnożenia 75
5.7 Pętle warunkowe: BEGIN...UNTIL i BEGIN .....
.....WHILE...REPEAT 77
5.8 Przykład: liczby pierwsze 81
Rozdział 6 PAMIĘĆ I EKRAN..... 84
6.1 Operacje na blokach pamięci..... 84
6.2 Wprowadzanie i wyprowadzanie ciągów znaków 85
6.3 Wprowadzanie liczb..... 88
Rozdział 7 JAK ZBUDOWANY JEST FORTH?..... 90
7.1 Kilka uwag wstępnych..... 90
7.2 Mapa pamięci 91
7.3 Główne części składowe 93
CZĘŚĆ II ZASTOSOWANIA 97
Rozdział 8 KOMPILACJA 99
8.1 Podsłowniki 99
8.2 Słowa natychmiastowe 101
8.3 Słowa tworzące rodziny słów 103
8.4 Definiowanie w kodzie maszynowym 106
8.5 Rekurencja 107

8.6.3 Wykonywanie 109
8.7 Kontrola błędów 112
8.8 Inicjalizacja 113
Rozdział 9 LICZBY RAZ JBSZCZE..... 114
9.1 Tablice 114
9.2 Formatowanie liczb 117
9.3 Jeszcze o liczbach podwójnych..... 120
9.4 Arytmetyka zmiennopozycyjna..... 120
9.5 Liczby losowe 121
Rozdział 10 AND, OH i XOR..... 123
10.1 Operatory logiczne i ich działanie 123
10.2 Maski bitowe..... 124
10.3 Bity ułatwiają wyszukiwanie 126
10.4 Przykład: rozpoznajmy rasy psów 127
Rozdział 11 NARZĘDZIA..... 133
11.1 Jak zmagazynować program ? 133
11.2 Edytor..... 131
11.3 Asembler 139
11.4 Grafika, dźwięk, programy użytkowe 140
Rozdział 12 PROGRAMY..... 141
12.1 Wieże z Hanoi..... 141
12.2 Osiem hetmanów 143
12.3 Gra Life 147
12.4 Liczby zespolone 153
12.5 Funkcje trygonometryczne 155
12.6 Sortowanie 156
12.6.1 Sortowanie przez przestawianie 157
12.6.2 Sortowanie pęcherzykowe 159
12.6.3 Sortowanie szybkie..... 160
ANEKSY..... 168
A.1 Bibliografia..... 166
A.2 Pełny edytor FORTHA 167
A.3 FORTH na Atari..... 176
A.4 Komunikaty błędów..... 181
A.6 Leksykon słów FORTHA..... 182
Spis treści..... 219

Cena zł

Wydawca SOETO
ul Hoża 50 00-682 Warszawa
tel. 21-64-01 w. 66
tlx. 894786