mikroKLAN 7

7 1984

informatyka

Technika mikroprocesorowa
-oprogramowanie

WYDAWNICTWO SIGMA

informatyka

7 1984

Nr 7
Miesięcznik Rok XIX
Lipiec 1984

Organ Komitetu Informatyki MNSzWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 14/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 2147. Obj. 4,0 ark. druk. Nakład 4200 egz. T-23.

INDEKS 36124

Cena egzemplarza zł 75,—
Prenumerata roczna zł 900,—

NACZELNA ORGANIZACJA TECHNICZNA
WYDAWNICTWO
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Roman Żelazny o narzędziach inżynierii oprogramowania
- Andrzej Szałas o systemach ekspertowych
- Ewa Gutman o bazie danych, obsługiwanej przez komputer pośredniczący
- Władysław Udrycki o języku CHILL
- Jerzy Szyller o mikroprocesorach lat osiemdziesiątych
- Józef de Mezer o sterowaniu alfaskopem
- Wacław Iszkowski o języku ojczystym informatyków

Technika mikroprocesorowa — czego nam brak?

Z pewnością brakuje nam oprogramowania mikrokomputerów, co widać gołym okiem, choćby po zawartości INFORMATYKI. Jak dotąd, niewiele miejsca na naszych łamach poświęcano tej tematyce. W bieżącym numerze podejmujemy pierwszą próbę zgrupowania materiałów dotyczących używanego w kraju oprogramowania systemów mikrokomputerowych.

Oczywiście, oprogramowanie mikrokomputerów obejmuje bardzo szeroki zakres produktów — od programów najprostszych gier do generatorów programów. W tym numerze położono nacisk raczej na oprogramowanie podstawowe, pomijając użytkowe. Dobre oprogramowanie podstawowe powinno być przede wszystkim doskonale dostosowane do sprzętu, wszechstronne i łatwe w użyciu. Prawdopodobnie według tych kryteriów wyodrębniły się pewne odróżnialne rodzaje oprogramowania mikrokomputerów, ściśle powiązane i przenikające się nawzajem, lecz mające szereg specyficznych cech — systemy operacyjne, kompilatory i narzędzia programowe.

Łatwe programowanie umożliwia przede wszystkim język wysokiego poziomu. Stwierdzenie, że językami asemblera nie podbije się świata ani nie upowszechni informatyki, choć przy ich użyciu można zrealizować dowolny algorytm, jest znane od dawna. Taka też była intencja powstania języka PL/M — zastąpienie języka asemblera dla 8-bitowych mikrokomputerów firmy INTEL. Językiem całkowicie odmiennym i konkurencyjnym dla PL/M jest FORTH — z pewnością trudniejszy do opanowania, ale zdobywający popularność znacznie szybciej od innych. W bieżącym numerze nie ma mowy o BASICU — najbardziej popularnym języku dla mikrokomputerów, lecz drukowaliśmy już artykuł na ten temat (INFORMATYKA, nr 5, 6, 7—8, 1983), a niebawem napiszemy o jego rozwoju pod wpływem techniki mikroprocesorowej (nawiasem mówiąc — naukę BASICA dla nowicjuszy rozpoczęły nawet HORYZONTY TECHNIKI, nr 2, 4, 1984). Jeżeli do BASICA dodamy PASCAL, to tendencja do wyeliminowania języków asemblera będzie wyraziście zilustrowana. Jednakże żaden z obecnie używanych translatorów wymienionych języków dla mikrokomputerów nie powstał w Polsce A czy mógł powstać?

Prace nad językiem PL/M rozpoczęła mała firma MICROCOMPUTER APPLICATIONS ASSOCIATES już w 1972 roku, na zlecenie INTELA; pierwszy znany mi artykuł opublikował Cary Kildalle dwa lata później (ELECTRONICS, Vol. 47, No. 13, p. 103, 27 June 1974). FORTH także nie spadł z nieba. Jego autor, Charles Moore, stopniowo ulepszał swą koncepcję — od początku lat sześćdziesiątych! Dopiero jednak w 1971 roku pojawiła się druga osoba programująca w tym języku i w tymże roku, w National Radio Astronomy Observatory (Kitt Peak, stan Arizona), powstał pierwszy rozpowszechniany system, aczkolwiek nie dla mikrokomputerów; pierwszy artykuł na ten temat opublikowano po dwóch latach (Proceedings of the IEEE, Vol. 61, No. 9, p. 1346, September 1973). Jeżeli więc trzeba dziesięciu lat użytkowania, aby język zrobił karierę i miał tak silną pozycję, to nie powinno nikogo dziwić, że w Polsce nie działają rodzime kompilatory nie tylko tych języków — nam brakuje konsekwencji w działaniu.

Podobnie jest z systemami operacyjnymi, które w bieżącym numerze reprezentuje CP/M (obecnymi artykułami kontynuujemy tę tematykę, rozpoczętą w ubiegłym roku). Proste monitory, stosowane w większości systemów mikrokomputerowych dotychczas omawianych w INFORMATYCE, to o wiele za mało, aby zapewnić dobre sprzężenie ze sprzętem, a zatem dobre jego wykorzystanie. Nowoczesny system operacyjny powinien spełniać szereg ogólnych warunków w zasadzie niemożliwych do pogodzenia i stąd bierze się podział tych systemów na dwie kategorie: systemy do wytwarzania oprogramowania, do których zalicza się CP/M, a także XENIX (oparty na UNIXIE), oraz — przeznaczone do pracy w czasie rzeczywistym, jak intelowski iRMX. System operacyjny CP/M powstał w roku 1974, a napisano go właśnie w języku PL/M, którego zresztą firma INTEL użyła do napisania pierwszej wersji własnego systemu operacyjnego RMX. Szacuje się, że obecnie CP/M lub jego odmiany są używane na kilkuset tysiącach instalacji i w kilku tysiącach konfiguracji. Stąd też problem, w jakim stopniu niezależny i oryginalny powinien być polski system operacyjny i czy jego powstanie ma sens.

Dobry system operacyjny i kompilator języka to bardzo dużo, ale o wiele za mało do efektywnej pracy. Do tego celu potrzebne jest wszechstronne środowisko programowania, a więc poza programami umożliwiającymi pisanie (edytor), tłumaczenie (asembler) i testowanie (program uruchomieniowy) innych programów, czego na ogół nam nie brak i to w różnych odmianach, powinno istnieć wiele różnych programów usługowych, np. umożliwiających dokumentowanie programów, obsługę urządzeń graficznych itd.; tych zaś w Polsce nigdy nie było. Ogólnie mówiąc — brakuje nam narzędzi programowych. Tu znów wzorem jest CP/M. Najnowszy katalog zawiera opis 2000 pakietów pracujących pod nadzorem tego systemu operacyjnego (CP/M Software Finder, 3rd edition, Digital Research, Pacific Grove, CA, November 1983). Inna sprawa, że takie oprogramowanie może powstać tylko w oparciu o sprawny system operacyjny, dysponujący przede wszystkim dobrym systemem plików.

Myśląc o przyszłości warto jednak pamiętać, że CP/M jest systemem jednozadaniowym i jednoużytkowym. Współczesne tendencje wyrażające się powstawaniem wieloprocesorowych zestawów mikrokomputerowych mogą doprowadzić do gwałtownej zmiany naszych wyobrażeń o programowaniu. Wiadomo jedynie, jakie cechy powinny mieć języki do programowania systemów wieloprocesorowych (po pierwszych doświadczeniach z ADA, CHILLEM, MODULĄ-2 itp.), trochę gorzej jest z rozpoznaniem właściwości systemów operacyjnych (choć powstały już ich odmiany, jak np. MP/M, CP/NET i współbieżny CP/M), a nie wiadomo niemal niczego o środowisku do programowania takich systemów, czyli — o odpowiednich narzędziach (np. o narzędziu do alokacji zadań na poszczególne procesory).

O ile wiemy do czego dąży się w tej dziedzinie za granicą, to o pracach krajowych nie da się wiele powiedzieć. Może da się powiedzieć tylko czego nam brak. Nie ma w kraju silnego ośrodka zajmującego się oprogramowaniem mikrokomputerów; są małe grupki — bo nawet nie zespoły — entuzjastów. Gdyby mnie ktoś zapytał, czy potrzebne jest w Polsce powstanie instytucji w rodzaju „Centrum Produkcji Oprogramowania", to bez wahania odpowiedziałbym twierdząco. Ale gdyby następne pytanie brzmiało, czy wierzę w skuteczność jego działania, to odpowiedziałbym, że nie.

Wydaje się, że planowe działanie dotyczące rozwoju krajowego oprogramowania mikrokomputerów (por. J. Dańda, INFORMATYKA, nr 1, 1983), które zresztą nigdy nie nabrały dużego rozmachu, ulegają z różnych przyczyn zahamowaniu. Przykładowo, opracowany kompilator skrośny MODULI nie uzyskał powodzenia, biblioteka B.ITE.M (ZETO Łódź) nie prosperuje najlepiej, a przygotowanie użytkowników do rozpoczęcia własnych prac programistycznych jedynie przez udział w Szkołach Mikroprocesorowych, to o wiele za mało. Potrzebny jest nam większy wysiłek.

Tak jak w przeszłości nie udawało nam się wytwarzanie komputera zbudowanego według oryginalnego pomysłu, tak też trudno będzie pisać oprogramowania według własnego patentu. Obserwując rozwój sytuacji w tej dziedzinie, sądzę, że rozstrzygające może być zdanie, które wypowiedział Gary Kildalle w odniesieniu do swojego CP/M: „Udoskonalenia? Przyjacielu, one zależą od ciebie" („Refinements? My friend, they're up to you").

JANUSZ ZALEWSKI

JERZY DAŃDA
ZBIGNIEW POZNAŃSKI
Instytut Technologii Elektronowej NPCP
Warszawa

Programowanie w języku PL/M (1)

Język PL/M był pierwszym językiem wysokiego poziomu przeznaczonym dla mikrokomputerów. Podobno zdefiniował go Gary Kildalle, który wykonał tę pracę dla firmy INTEL. Ponieważ jest on znanym entuzjastą języka PL/I — wziął za podstawę ten właśnie język.

PL/M musiał być językiem prostym, bo w owych czasach (połowa lat siedemdziesiątych) na ogół mikrokomputery programowało się w języku asemblera. Dlatego też jest on bardzo krytykowany. Szczególnie irytujący jest brak liczb zmiennoprzecinkowych.

Implementacja języka PL/M na mikrokomputery 8-bitowe (PL/M-80 dla mikroprocesora 8080) przyjęła się nieźle. Firmy konkurencyjne (MOTOROLA, ZILOG) ze względu na konieczność dotrzymania kroku firmie INTEL zostały zmuszone do wprowadzenia własnych mutacji języka PL/M.

Wraz z mikroprocesorem 8086, INTEL wprowadził rozszerzony język PL/M, a jego implementacja, PL/M-86, jest już pozbawiona wcześniejszych niedoskonałości. Prócz tego INTEL dostarcza — bardzo drogi i złożony — program wykonujący tekstowe przetwarzanie programów napisanych w języku PL/M-80 na język PL/M-86. Chociaż, jak wiadomo, takie przetwarzanie często nie może być wykonane automatycznie w 100% — to jednak wspomniany program (podobno wykorzystujący niektóre osiągnięcia sztucznej inteligencji) bardzo dobrze wspomaga programistę w realizacji tego zadania. Produkt ten nie jest jeszcze, niestety, dostępny w Polsce.

Program w języku PL/M może się składać z jednego lub kilku modułów, przy czym jeden z nich musi być modułem głównym. Modułem jest etykietowany blok DO, który nie jest zawarty w żadnym innym bloku, np.:

ALGORYTM$PID: DO;
<deklaracje>
<instrukcje>
END ALGORYTM$PID;

Tylko w module głównym oprócz deklaracji występują także instrukcje. Każdy z modułów jest kompilowany oddzielnie. Po kompilacji moduł może być łączony z innymi modułami tworzącymi program. Wszystkie zmienne, tablice, struktury i procedury zadeklarowane w danym module są w nim lokalne, tzn. dostępne tylko w tym module. Wyjątek stanowią deklaracje z atrybutami PUBLIC i EXTERNAL, rozszerzającymi zakres tych deklaracji na inne moduły. Dzięki tym atrybutom moduł nie stanowi zamkniętej jednostki, lecz może wymieniać informacje z innymi modułami. Koncepcja modułu ma duże znaczenie przy tworzeniu większych programów, które mogą być pisane i uruchamiane przez różne osoby. Ponadto uzyskuje się większą przejrzystość oprogramowania i znaczne skrócenie czasu jego uruchomienia.

Poniżej omówimy kolejno poszczególne konstrukcje języka PL/M — od podstawowych do najbardziej złożonych. Sposób ich przedstawienia jest jednak dość nietypowy. Dla ułatwienia posługiwania się językiem, użycie wszystkich jego elementów wyjaśniamy rozpoczynając od przykładów.

PRZYKŁAD 1 — OBLICZANIE WARTOŚCI FUNKCJI

Moduł przedstawiony na wydruku 1 jest modułem głównym, ponieważ oprócz deklaracji zawiera ciąg instrukcji. Każda deklaracja zaczyna się od słowa DECLARE, po którym wymienia się jedną bądź więcej nazw zmiennych. Słowo DECLARE jest słowem zastrzeżonym, którego nie można użyć w programie w innym kontekście. Pierwsza deklaracja przypisuje identyfikatorowi **B1** cyfrę 1, zaś identyfikatorowi **B4** — cyfrę 4. Każde wystąpienie tych nazw w programie powoduje wstawienie w ich miejsce tekstu zawartego w apostrofach po słowie LITERALLY. Jest to tzw. deklaracja synonimu nazwy. Następnie zadeklarowano w programie zmienne dwóch typów — **BYTE (M)** oraz **ADDRESS (M$PREV,K,TS,TI,TD,Q)**. Zmienne typu BYTE zajmują 1 bajt pamięci, zaś typu ADDRESS — 2 bajty. Deklarując listę zmiennych tego samego typu można je podać w nawiasie.

```
KOMPENSACJA: DO;

DECLARE B1 LITERALLY '1', B4 LITERALLY '4';
DECLARE (M$PREV,Q) ADDRESS INITIAL(300,0);
DECLARE (K,TS,TI,TD) ADDRESS DATA(10,100,450,450);

PRZEPLYW: PROCEDURE(MP);
DECLARE MP BYTE;
  OUTPUT(B4) = MP;
  Q = Q + TS*(K*(M$PREV-Q)/TI + TD*K*(MP - M$PREV)/TI;
END PRZEPLYW;

CALL PRZEPLYW(INPUT(B1));

END KOMPENSACJA;
```

Wydruk 1. Program modułu 1

Pierwsze instrukcje programu przypisują poszczególnym zmiennym wartości stałe (dziesiętne lub szesnastkowe). Przy wielokrotnym przypisaniu nazwy zmiennych oddziela się przecinkiem. Kolejna instrukcja, **M=INPUT (B1)**, przypisuje zmiennej **M** wartość systemowo zdefiniowanej funkcji INPUT, która odczytuje 1 bajt danych z portu o numerze określonym przez parametr. Przy użyciu tablicy OUTPUT wartość wczytana z portu 1 zostaje wyprowadzona do portu określonego numerem elementu tej tablicy. Ostatnia instrukcja programu oblicza wartość funkcji dla wartości zmiennej **M**, odczytanej z portu 1, a wynik tego obliczenia przypisuje zmiennej **Q.** .

Zmienne i stałe

Zmienne występujące w programie mogą być zmiennymi prostymi, tablicami lub strukturami. Zmienna prosta, zwana dalej zmienną, ma jedną wartość liczbową mogącą ulegać zmianie podczas wykonywania programu. Nazwa zmiennej zaczyna się od litery, po której mogą następować litery lub cyfry. W celu poprawienia czytelności programu, stosuje się znak dolara, pomijany przez kompilator, np. **M$PREV, ZMIENNA$STERUJĄCA, PARAMETR$WEJŚCIOWY** itp. W deklaracji zmiennej należy podać jej typ. W języku PL/M zmienne mogą być typu BYTE lub ADDRESS. Zmienne typu BYTE zajmują 8 bitów pamięci (1 bajt), zaś zmienne typu ADDRESS — 16 bitów (2 bajty). Tak więc zakres liczb typu BYTE obejmuje przedział 0—255, a liczb typu ADDRESS — 0—65535. Rezerwowanie pamięci dla zmiennych odbywa się w kolejności ich zadeklarowania.

Stała w języku PL/M może być liczbą lub ciągiem znaków. Stałe liczbowe mają jedną z następujących postaci:

- dwójkową, złożoną z cyfr **0,1**, zakończoną literą **B**, np. **00001101B, 0000000100101100B**
- ósemkową, złożoną z cyfr **0—7**, zakończoną literą **O**, np. **15O, 454O**
- dziesiętną, złożoną z cyfr **0—9**, zakończoną literą **D** (którą można pominąć), np. **13, 300**

● szesnastkową, złożoną z cyfr **0—9** i liter **A—F**, zakończoną literą **H** (liczby zaczynające się od litery poprzedza się zerem), np. **0DH, 12CH.**

W celu zwiększenia czytelności programu, między znaki stałych można wstawić znak dolara, np. **0000$0001$0010$ $1100B.** W przykładzie 1 wykorzystano najczęściej stosowane postacie stałych, tj. dziesiętną (stałe 13,300,0) oraz szesnastkową (stałe 1CH,45H).

Wyrażenia i operatory arytmetyczne

Wyrażeniem nazywamy formułę lub regułę obliczenia, która wyznacza pewną wartość. Wyrażenie składa się z argumentów i operatorów. Argumentami są stałe, zmienne lub wartości przyjmowane przez funkcje. Operatory dzielą się na arytmetyczne, logiczne i relacyjne. W tym przykładzie — do obliczenia wartości funkcji wykorzystano wyrażenie zawierające tylko operatory arytmetyczne. Inne operatory wystąpią w następnych przykładach.

W języku PL/M istnieje pięć podstawowych operatorów arytmetycznych: dodawanie +, odejmowanie —, mnożenie *, dzielenie /, modulo **MOD**. Wszelkie operacje arytmetyczne są wykonywane na liczbach traktowanych jako całkowite bez znaku. Przykładowo — reprezentacja liczb —5 i 251 jest jednakowa i w zapisie szesnastkowym ma postać 0FBH. Wynik dodawania i odejmowania argumentów typu BYTE jest umieszczany w jednym bajcie pamięci, zaś typu ADDRESS — w dwóch bajtach. Jeśli jeden z argumentów jest typu ADDRESS, drugi typu BYTE, to argument jednobajtowy jest rozszerzany do dwóch bajtów i operacja jest wykonywana w arytmetyce dwubajtowej. Mnożenie i dzielenie zawsze daje wynik w dwóch bajtach pamięci. W przypadku, gdy wskutek wykonania operacji arytmetycznej wystąpi przekroczenie zakresu liczb, bity najbardziej znaczące są usuwane bez sygnalizacji. Wynik dzielenia jest zawsze zaokrąglany z niedomiarem do liczby całkowitej, zaś dzielenie przez zero jest nieokreślone. Operator odejmowania może być także wykorzystany jako operator jednoargumentowy, np. —5; wynikiem działania tego operatora na liczbę jest jej uzupełnienie dwójkowe, np. —5→251. Operator modulo (MOD) omówimy w bardziej złożonych przykładach. Kolejność działania operatorów arytmetycznych jest typowa.

Pozostałe konstrukcje użyte w przykładzie

Najprostszą instrukcją w języku PL/M jest instrukcja przypisania o postaci:

zmienna = wyrażenie

Powoduje ona nadanie zmiennej wartości wyrażenia występującego z prawej strony znaku =. Ostatnia instrukcja przypisania w tym przykładzie zawiera złożone wyrażenie arytmetyczne, którego wartość zostaje przypisana zmiennej **Q**. Jeżeli zachodzi konieczność nadania tej samej wartości wielu zmiennym, to nazwy tych zmiennych podaje się z lewej strony instrukcji przypisania, oddzielając je przecinkiem, np.:

ZMIENNA$1,ZMIENNA$2,ZMIENNA$3 = 0;

Jeżeli wartość wyrażenia w instrukcji przypisania jest typu BYTE, a zmienna — typu ADDRESS, to obliczona wartość wyrażenia zostaje rozszerzona do dwóch bajtów przez wypełnienie ośmiu bardziej znaczących bitów zerami. Kiedy zmiennej typu BYTE przypisywana jest wartość dwubajtowa (ADDRESS), to danej zmiennej zostanie przypisanych osiem mniej znaczących bitów, a osiem bardziej znaczących bitów zostanie pominiętych.

Deklaracja z wykorzystaniem słowa LITERALLY o postaci:

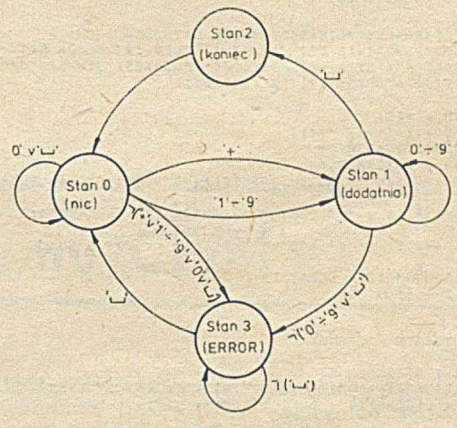
DECLARE<nazwa synonimu>LITERALLY<'tekst zastępowany'>;

oznacza, że każdorazowe użycie nazwy synonimu powoduje zastąpienie go tekstem zawartym w apostrofach. Stosowanie synonimów nazw jest szczególnie wygodne wtedy, gdy zastępowany tekst pojawia się często i często zachodzi potrzeba jego zmiany. W tym ostatnim przypadku wystarczy tylko zmienić tekst w deklaracji synonimu, bez wprowadzania jakichkolwiek zmian w pozostałej części programu.

Funkcja INPUT i tablica OUTPUT służą do komunikacji programu z portami mikroprocesora 8080 i są zdefiniowane systemowo. INPUT jest funkcją typu BYTE o wartości odczytanej z portu o numerze określonym przez parametr aktualny. Parametrem może być tylko stała liczbowa z przedziału 0—255. Tablica OUTPUT, której każdy element jest typu BYTE, zaś indeks odpowiada numerowi portu wyjściowego mikroprocesora, służy do wyprowadzania bajtu danych do odpowiedniego portu, np. **OUTPUT (B4)=M**; Odwołanie do elementów tej tablicy musi wystąpić z lewej strony instrukcji przypisania.

PRZYKŁAD 2 — ROZPOZNAWANIE ZNAKÓW

Zakładamy, że w buforze wejściowym znajduje się ciąg znaków w kodzie ASCII. Automat rozpoznaje liczby dodatnie (ze znakiem lub bez), dokonuje konwersji rozpoznanej liczby na kod dwójkowy i umieszcza ją w buforze wyjściowym. Za liczbę uznaje się ciąg cyfr zakończony spacją. Jeśli liczbę poprzedza znak +, to dopuszcza się zera pomiędzy tym znakiem a pierwszą cyfrą różną od zera, np. +025. Napotkanie każdego innego znaku w ciągu cyfr oznacza, że liczba jest błędna. Po błędnym znaku, każdy nowy znak różny od spacji jest rozpoznawany jako błąd. Graf przejść automatu rozpoznającego przedstawiono na rysunku.



Graf przejść automatu rozpoznającego

Program realizujący powyższy algorytm, przedstawiony na wydruku 2, stanowi moduł główny. Zadeklarowano w nim synonimy LIT oraz DCL dla słów LITERALLY i DECLARE, dzięki czemu skrócono deklaracje pozostałych synonimów. Następnie zadeklarowano dwie tablice **BUFOR$WE** oraz **BUFOR$WY**, podając ich wymiary i typy elementów. Pierwsza tablica stanowi bufor wejściowy automatu. Jej elementami są znaki w kodzie ASCII zajmujące po jednym bajcie pamięci. Tablica **BUFOR$WY** jest buforem wyjściowym, do którego przesyła się rozpoznane liczby. Ponieważ liczby mogą być większe od 255, elementy tablicy muszą być dwubajtowe (typu ADDRESS). Trzy zmienne **LICZBA,I,J** są zadeklarowane i jednocześnie zainicjowane. Podanie w deklaracji słowa INITIAL powoduje że podczas ładowania programu zmiennym tym przypisywane są wartości podane w nawiasie. W podobny sposób można inicjować elementy tablic i struktur.

Ciąg instrukcji zawartych w bloku DO WHILE jest wykonywany do chwili napotkania w buforze wejściowym **BUFOR$WE** znaku #. Istotą działania automatu jest instrukcja DO CASE. W zależności od wartości wyrażenia znajdującego się za słowem CASE, wykonywana jest odpowiednia instrukcja wewnętrzna bloku (każdej liczbie naturalnej stanowiącej wartość zmiennej **STAN$AUTOMATU** odpowiada określona instrukcja). Kolejne instrukcje bloku DO CASE opisują zachowanie automatu w czterech określonych stanach. Zerowa instrukcja bloku, odpowiadająca stanowi **0** automatu, jest instrukcją warunkową typu IF..THEN..ELSE. Jeżeli znak w buforze wejściowym jest spacją (**20H** w kodzie ASCII) lub zerem (**'0'**), to automat pozostaje w stanie zerowym. Jeżeli rozpoznano znak '+' lub znak z przedziału '1'—'9', to zgodnie z rysunkiem automat przechodzi do stanu pierwszego, przy czym w tym drugim przypadku zmniejsza o 1 indeks **I** tablicy **BUFOR$WE**. Ta dodatkowa operacja jest konieczna z uwagi na uwzględnianie rozpoznanego znaku z przedziału '1'—'9' w stanie pierwszym automatu, w którym dokonuje się konwersji znaku. Po wyjściu z bloku DO CASE indeks tablicy **BUFOR$WE** jest zwiększany, co jest równoznaczne

z przejściem do analizy następnego znaku. Jeżeli w buforze wejściowym znajduje się znak nielegalny, to automat przechodzi do stanu trzeciego (ERROR).

Pierwsza instrukcja bloku DO CASE opisuje zachowanie automatu w stanie pierwszym, tzn. gdy w buforze wejściowym wykryty został element numeryczny. Jeżeli znak w buforze wejściowym należy do przedziału **'0'—'9'**, to oblicza się wartość rozpoznawanej liczby według algorytmu

LICZBA = LICZBA*10 + odczytana cyfra (dwójkowo).

Ponieważ odczytana cyfra zapisana jest w kodzie ASCII, konieczna jest jej konwersja do postaci dwójkowej, co realizuje prosty algorytm:

odczytana cyfra(binarnie) = odczytana cyfra(ASCII) — '0'.

Jeżeli rozpoznano znak spacji, oznaczający koniec elementu numerycznego, to następuje przejście automatu do stanu drugiego (KONIEC). Napotkanie każdego innego znaku zmienia stan automatu na trzeci (ERROR) z jednoczesnym wyzerowaniem zmiennej LICZBA (nielegalny element numeryczny).

```
AUTOMAT$ROZPOZNAJACY: DO;

DECLARE LIT LITERALLY 'LITERALLY', DCL LIT 'DECLARE';
DCL NIC LIT '0', DODATNIA LIT '1', KONIEC LIT '2', ERROR LIT '3';
DCL BUFOR$WE(0FFFH) BYTE, BUFOR$WY(200) ADDRESS, STAN$AUTOMATU BYTE;
DCL (LICZBA,I,J) ADDRESS INITIAL(0,1,0);

STAN$AUTOMATU = NIC;
DO WHILE BUFOR$WE(I) <> '/';
  DO CASE STAN$AUTOMATU;
     /* STAN 0 AUTOMATU */
     IF BUFOR$WE(I)=' ' OR BUFOR$WE(I)='0' THEN STAN$AUTOMATU = NIC;
     ELSE
     IF BUFOR$WE(I) = '+' THEN STAN$AUTOMATU = DODATNIA;
     ELSE
     IF BUFOR$WE(I) > '0' AND BUFOR$WE(I) <= '9' THEN
     DO;
        STAN$AUTOMATU = DODATNIA;
        I = I - 1;
     END;
     ELSE
     STAN$AUTOMATU = ERROR;

     /* STAN 1 AUTOMATU */
     IF BUFOR$WE(I) >= '0' AND BUFOR$WE(I) <= '9' THEN
     LICZBA = LICZBA*10 + BUFOR$WE(I) - '0';
     ELSE
     IF BUFOR$WE(I) = ' ' THEN
     DO;
        STAN$AUTOMATU = KONIEC;
        I = I - 1;
     END;
     ELSE
     DO;
        STAN$AUTOMATU = ERROR;
        LICZBA = 0;
     END;

     /* STAN 2 AUTOMATU */
     DO;
        BUFOR$WY(J) = LICZBA;
        LICZBA = 0;
        J = J + 1;
        STAN$AUTOMATU = NIC;
     END;

     /* STAN 3 AUTOMATU */
     IF NOT BUFOR$WE(I) = ' ' THEN STAN$AUTOMATU = ERROR;
     ELSE
     STAN$AUTOMATU = NIC;
  END;
  I = I + 1;
END;

END AUTOMAT$ROZPOZNAJACY;
```

Wydruk 2. Program modułu 2

W stanie drugim automatu, reprezentującym fakt rozpoznania pełnego elementu numerycznego, wartość zmiennej LICZBA jest przesyłana do bufora wyjściowego. Zmienna ta przyjmuje następnie wartość zero, a automat ponownie przechodzi do stanu zerowego w celu rozpoznania nowej liczby. Zachowanie automatu w stanie drugim jest opisane kilkoma instrukcjami zawartymi w bloku DO..END. Trzeci (ERROR) stan automatu opisany jest instrukcją warunkową IF..THEN..ELSE. W stanie tym automat pozostaje do chwili wykrycia znaku różnego od spacji. Po zakończeniu programu w tablicy **BUFOR$WY** znajdą się wszystkie rozpoznane liczby.

Tablice

W deklaracji tablicy należy podać jej nazwę, wymiar (stała ujęta w nawiasy) oraz typ elementów, np.

DECLARE BUFOR$WE(0FFFH) BYTE;

W powyższym przykładzie elementami tablicy są dane typu BYTE, dla których zarezerwowano 0FFFH bajtów pamięci. Odwoływanie się do poszczególnych elementów tablicy następuje przez podanie jej nazwy oraz indeksu ujętego w nawiasy. Jako indeks może być użyta stała liczbowa, zmienna lub wyrażenie, którego wartość określa odpowiedni element tablicy. Elementy tablicy są zawsze numerowane od zera, tzn. zbiór indeksów zawiera zero, natomiast podając wymiar tablicy liczymy jej elementy poczynając od 1. Deklaracja kilku tablic z jednakowym wymiarem oraz typem elementów może mieć następującą postać:

DECLARE (BUF1,BUF2,BUF3) (100) ADDRESS;

Tablice odgrywają szczególną rolę w instrukcjach iteracyjnych, gdzie możliwa jest automatyczna zmiana indeksu. Pozwala to na łatwy dostęp do wszystkich elementów tablicy. W tym przykładzie instrukcję taką stanowi blok DO WHILE, w którym zmienny jest indeks **I** tablicy **BUFOR$WE**, dzięki czemu można odczytywać kolejne wartości elementów tej tablicy, analizować je lub dokonywać różnych operacji.

Operatory relacyjne i logiczne

Operatory relacyjne w języku PL/M są następujące: = równy, >= większy lub równy, > większy, <= mniejszy lub równy, < mniejszy, <> różny. Mogą działać na argumentach typu BYTE lub ADDRESS. Argumenty są traktowane jako liczby całkowite bez znaku. W wyniku otrzymuje się wielkość BYTE z wartością **0FFH** (TRUE), jeżeli warunek jest spełniony, lub **00H** (FALSE) w przypadku niespełnienia warunku. W tym przykładzie wielokrotnie wykorzystano operatory relacyjne w celu identyfikacji odpowiednich znaków.

W języku PL/M wyróżnia się cztery operatory logiczne: NOT, AND, OR i XOR. Każdy z nich wykonuje operacje logiczne na danych jednobajtowych i dwubajtowych. Najprostszym operatorem jest NOT (negacja). Operator AND jest często wykorzystywany do maskowania bitów jednego z argumentów. W tym przykładzie operatory AND i OR wykorzystano do identyfikacji cyfry oraz znaku spacji lub zera. Kolejność działań poszczególnych grup operatorów (w wyrażeniach bez nawiasów) jest następująca: arytmetyczne, relacyjne, logiczne.

Instrukcje złożone

W języku PL/M instrukcje mogą być łączone w instrukcje złożone (bloki) DO..END o następującej postaci:

```
DO;
   instrukcja 1;
   instrukcja 2;
   . . . . . .
   instrukcja N;
END;
```

Blok DO..END jest traktowany jako jedna instrukcja i może występować w każdym miejscu programu. Najczęściej wykorzystuje się go w instrukcjach warunkowych IF..THEN..ELSE i DO CASE. Ponieważ pojedynczy blok DO..END stanowi jedną instrukcję, istnieje możliwość zagnieżdżania bloków. Wystarczy — przykładowo — zastąpić instrukcję 2 (w ogólnej postaci bloku DO..END) nowym blokiem DO..END.

Instrukcja iteracyjna (blok) DO WHILE umożliwia wykonywanie obliczeń w pętli i ma następującą postać:

```
DO WHILE<wyrażenie>;
   instrukcja 1;
   instrukcja 2;
   . . . . . .
   instrukcja N;
END;
```

Działanie bloku DO WHILE jest następujące:

- Obliczana jest wartość wyrażenia występującego po słowach DO WHILE. Jeśli najmniej znaczący bit obliczonej wartości wyrażenia jest równy jedności, wtedy wykonywane są instrukcje wewnątrz bloku DO WHILE.
- Po wykonaniu całego ciągu instrukcji bloku, ponownie obliczana jest wartość wyrażenia. Instrukcje wewnątrz bloku DO WHILE są wykonywane w pętli do chwili, gdy najmniej znaczący bit (**NZB**) wartości wyrażenia przyjmie wartość równą zero. Wtedy wykonywana jest pierwsza instrukcja występująca za słowem END tego bloku.

Jeżeli zachodzi konieczność wykonania jednej z wielu instrukcji w zależności od wartości zmiennej lub wyrażenia, to wygodnie jest wykorzystać instrukcję (blok) DO CASE, która ma następującą postać:

```
DO CASE<wyrażenie>;
   instrukcja 0;
   instrukcja 1;
   . . . . . .
   instrukcja N-1;
END;
```

Przy wejściu do tego bloku obliczana jest wartość wyrażenia występującego za słowem CASE, która powinna być liczbą naturalną z przedziału [0,N-1]. Kolejnym liczbom z tego przedziału przyporządkowane są kolejne instrukcje w bloku. W zależności od wartości wyrażenia wykonywana jest więc odpowiednia instrukcja wewnętrzna.

Instrukcję warunkową IF..THEN..ELSE wykorzystuje się wtedy, gdy zachodzi konieczność wykonania określonych działań w zależności od spełnienia pewnych warunków. Ogólna postać tej instrukcji jest następująca:

IF<wyrażenie>THEN instrukcja 1; ELSE instrukcja 2;

Wyrażenie występujące po słowie IF może być dowolne, przy czym najczęściej jest to wyrażenie warunkowe (logiczne, relacyjne). Jeśli najmniej znaczący bit obliczonej wartości tego wyrażenia jest równy 1 (TRUE), to wykonywana jest instrukcja 1, w przeciwnym zaś przypadku — instrukcja 2. Jeśli nie przewiduje się wykonania żadnej instrukcji, gdy najmniej znaczący bit wartości wyrażenia jest równy 0, to część ELSE można pominąć. Instrukcja ma wówczas postać:

IF<wyrażenie>THEN instrukcja 1;

Instrukcja warunkowa z tego przykładu, opisująca zachowanie automatu w stanie 0 jest dość złożona i dobrze ilustruje możliwości tworzenia skomplikowanych konstrukcji językowych w języku PL/M. Jej ogólna postać jest następująca:

IF<wyrażenie 1>THEN instrukcja 1; ELSE
IF<wyrażenie 2>THEN instrukcja 2; ELSE
IF<wyrażenie 3>THEN instrukcja 3; ELSE instrukcja 4;

JERZY DWORZECKI
Computer Studio Kajkowski
Gdynia

Biblioteka programów dla systemu CP/M

System operacyjny CP/M stał się już standardem dla mikrokomputerów opartych na mikroprocesorach INTEL 8080, 8085 i ZILOG Z80. Istnieją także wersje CP/M dla mikrokomputerów opartych na innych mikroprocesorach, np. SOFTCARD dla APPLE II, CP/M-86 dla INTELA 8086/ /8088 czy CP/M-68K dla MOTOROLI 68000.

Ocenia się, że obecnie ok. trzystu typów mikrokomputerów używa systemu CP/M jako podstawowego lub alternatywnego systemu operacyjnego, a ok. pięćdziesięciu firm dostarcza oprogramowanie zgodne z nim. Nie należy zapominać także o setkach tysięcy programistów, którzy tworzą i wymieniają oprogramowanie przez sieć klubów skupiających użytkowników CP/M na całym świecie.

W ciągu dziesięciu lat, które minęły od pojawienia się systemu CP/M, powstała ogromna biblioteka programów. Nie ma praktycznie problemu obliczeniowego, którego rozwiązania nie można by znaleźć (jako program lub pakiet programów) w tej bibliotece. Jeżeli istnieje potrzeba stworzenia indywidualnego oprogramowania, można wykorzystać istniejące w systemie CP/M narzędzia, które ułatwiają i przyspieszają programowanie.

Na podstawie obserwacji rynku i zapowiedzi producentów można stwierdzić, że także w Polsce CP/M lub inny system z nim zgodny stanie się podstawowym systemem operacyjnym dla mikrokomputerów. W ten sposób powstaje szansa wykorzystania gotowego oprogramowania, a więc uniknięcia niepotrzebnych nakładów i straty czasu. Aby jednak przyszły użytkownik mógł wybrać odpowiedni dla siebie program z biblioteki CP/M, musi poznać jej zawartość.

Celem poniższego artykułu jest przedstawienie wybranych programów i pakietów programowych z biblioteki CP/M. Omówione zostanie jedynie oprogramowanie profesjonalne, tzn. dostępne na rynku, z pominięciem tzw. zbioru PUBLIC DOMAIN, do którego uzyskuje się dostęp wstępując do klubu użytkowników CP/M. Nie będzie omawiane także oprogramowanie wchodzące w skład systemu CP/M — już opisane w INFORMATYCE [3].

Przeważająca część prezentowanych niżej programów jest wykorzystywana w firmie CSK, która od ponad dwóch lat zajmuje się systemem CP/M i systemami z nim zgodnymi.

JĘZYKI PROGRAMOWANIA

W systemie CP/M można tworzyć oprogramowanie praktycznie w każdym języku. Najbardziej popularny jest BASIC, a wśród różnych dialektów tego języka najbardziej rozpowszechnione są produkty firmy MICROSOFT — interpreter BASIC-80 (znany też jako MBASIC) oraz kompilator BASCOM. Dzięki popularności i wysiłkom tej firmy, BASIC-80 ma szansę stać się standardem dla mikrokomputerów 8-bitowych BASIC-80 umożliwia m.in.:

— używanie liczb pojedynczej i podwójnej precyzji (8 i 16 pozycji znaczących)

— programowanie strukturalne przy użyciu instrukcji IF..THEN..ELSE i WHILE..WEND

— redagowanie i testowanie programu (EDIT, TRACE, ON ERROR)

— przetwarzanie plików o organizacji sekwencyjnej i losowej

— wywoływanie podprogramów w języku maszynowym i przekazywania parametrów (CALL, USR, PEEK, POKE).

Duże znaczenie praktyczne ma możliwość pisania i testowania programu przy użyciu interpretera, a następnie — stworzenia skompilowanej postaci uruchomieniowej.

Produktom MICROSOFTU dorównują, pod względem popularności, translator CBASIC oraz kompilator CB-80 firmy DIGITAL RESEARCH. Dla translatora charakterystyczne jest tłumaczenie kodu źródłowego na kod pośredni (P-Code), interpretowany w czasie wykonywania programu. CBASIC i CB-80 umożliwiają operowanie liczbami rzeczywistymi zapisanymi w formacie BCD (14 pozycji znaczących), mają rozbudowaną grupę instrukcji do przetwarzania napisów (ang. strings) oraz przetwarzania plików.

Programiści, którzy stosują metody inżynierii oprogramowania i wymagają od języka bardziej nowoczesnych struktur, mają do dyspozycji S-BASIC, który zawiera zbliżone do stosowanych w PASCALU programowe struktury sterowania i umożliwia definiowanie procedur ze zmiennymi lokalnymi. Podobną rolę może spełniać STRUBAS — preprocesor, który umożliwia programowanie strukturalne.

Znane zalety PASCALA powodują, że jest on również bardzo popularny wśród użytkowników mikrokomputerów. W systemie CP/M można wykorzystywać m.in. kompilator JRT-PASCAL, który mimo niskiej ceny (29,95 dol.) może zadowolić nawet wymagających programistów. Ograniczenia możliwości wyboru w instrukcji CASE do 128 przypadów lub rozmiaru plików losowych do 8 M bajtów nie są zbyt istotne w praktyce. Rozszerzone możliwości w stosunku do Standard-PASCAL (wg ISO) ma PASCAL/MT+ firmy DIGITAL RESEARCH. Kompilator tego języka tłumaczy program bezpośrednio na relokowalny kod maszynowy. Rozszerzenie dotyczy instrukcji przetwarzania napisów (porównywanie, łączenie, przeszukiwanie napisów o długości do 255 znaków) oraz typów danych.

FORTRAN można stosować dzięki kompilatorowi FORTRAN-80 firmy MICROSOFT. Odpowiada on FORTRANOWI-IV wg normy ANSI-66 (z wyjątkiem typu COMPLEX). Jest to najbardziej rozpowszechniona wersja tego języka i brak takich instrukcji jak IF..THEN..ELSE..ENDIF może nie przeszkadzać doświadczonym programistom. Natomiast FORTRAN-80 zawiera typ BYTE oraz instrukcje PEEK i POKE.

COBOL nie jest tak popularny na mikrokomputerach, jak na dużych komputerach, jednak istnieje duża liczba kompilatorów tego języka w systemie CP/M. COBOL-80 firmy MICROSOFT pozwala na wykorzystanie konwersacyjnych możliwości mikrokomputerów, zapewnia dużą precyzję obliczeń (18 pozycji znaczących) oraz ułatwia programowanie grafiki (menu i maski ekranowe). CIS-COBOL firmy MICRO FOCUS jest zgodny z normą ANSI-74 i ma dodatkowe możliwości pracy konwersacyjnej, dzięki czemu można wykorzystywać cały ekran do wprowadzania i wyprowadzania danych, np.:

DISPLAY "Podaj nazwe artykulu" AT 1220 ACCEPT NAZWA AT 1248

gdzie 12 oznacza numer wiersza, a 20 i 48 numery kolumn.

Osoby, które chcą w pełni wykorzystać zalety programowania w języku maszynowym, mogą sięgnąć po MACRO-80. Pakiet składa się z makroasemblera dla mikroprocesorów 8080 i Z80, programu łączącego MS-LINK, programu zarządzającego biblioteką modułów MS-LIB oraz programu tworzącego tablice odwołań zewnętrznych (ang. cross reference table) MS-CREF. Moduły programowe utworzone za pomocą MACRO-80 mogą być łączone z modułami utworzonymi przez kompilatory FORTRAN-80, COBOL-80 i BASIC-80. Podobny zestaw narzędzi dostarcza firma DIGITAL RESEARCH. Nazwy programów wyjaśniają ich funkcje: RMAC, LINK-80, LIB, XREF.

W systemie CP/M można programować również w następujących językach: FORTH, LISP, PL/I, ADA, a także — w coraz popularniejszym języku C [1]. W tabeli podano wyniki testowania kompilatorów niektórych języków.

OPROGRAMOWANIE NARZĘDZIOWE

W tej grupie znajdują się programy ułatwiające pisanie, uruchamianie, testowanie i dokumentowanie innych programów oraz — operowanie w systemie CP/M.

Niezbyt wygodny w użyciu edytor systemowy ED.COM może być zastąpiony przez EDIT-80 (o możliwościach zbliżonych do edytora dla IBM 360/370), VEDIT — ekranowy edytor tekstowy lub — przez jeden z wielu programów przetwarzania tekstów. Do uruchamiania i testowania programów służy SID/ZSID (ang. symbolic instruction debugger). Zwiększa on możliwości standardowego programu DDT o mnemoniki Z80 i nazwy symboliczne. TRACE 80 dla mikroprocesora Z80 i TRACE 85 dla 8085 są najwszechstronniejszymi symbolicznymi programami uruchomieniowymi tych mikroprocesorów. Zawierają ponad 50 poleceń.

Inne programy ułatwiają obsługę plików dyskowych oraz zachowanie plików w przypadku awarii. DISK DOCTOR zachowuje zawartość pliku po poleceniu ERA i kopiuje dyski z uszkodzonymi sektorami. DPATCH umożliwia redagowanie zawartości dysku (przeglądanie, zmianę zawartości sektorów, przeszukiwanie sektorów w celu odnalezienia podanego ciągu znaków itp.) oraz wielokrotne odczytywanie błędnego sektora, np. przez najazd głowicy z różnych kierunków.

SYSTEM-CHECKER i DIAGNOSTICS II są pakietami do testowania zasobów sprzętowych mikrokomputera: procesora, pamięci, napędów dyskowych, terminali i drukarek. Powtarzanie plików użytkowych ułatwia ACCESS MANAGER. Zapewnia on dostęp losowy i indeksowo-sekwencyjny do plików w programach napisanych w językach CB-80, PL/I i PASCAL/MT+. Podobną rolę odgrywa MULTISAM dla kompilatorów firmy MICROSOFT.

Sortowanie danych umożliwia M/SORT (bez ograniczeń długości kluczy oraz długości i organizacji rekordów) oraz SUPERSORT, który sortuje do 32 plików, z rekordami o dowolnym formacie pól, przy zmiennych kryteriach doboru rekordów itp. Przydatnym narzędziem dla użytkowników systemu ISIS może być XCPM, który umożliwia jednoczesne tworzenie oprogramowania dla ISIS i CP/M, oraz ICL, który przenosi oprogramowanie z ISISA pod CP/M.

PAKIETY UŻYTKOWE

W bibliotece CP/M znajduje się oprogramowanie adresowane do finalnego użytkownika mikrokomputera, osoby nie przygotowanej zawodowo do pracy z komputerem. Jest to oprogramowanie uniwersalne, gotowe do natychmiastowego stosowania lub służące do szybkiego tworzenia specjalizowanych systemów przetwarzania danych. Powstanie i rozpowszechnienie się tego rodzaju oprogramowania jest ściśle związane z rozwojem mikrokomputerów. Znacznie polepsza ono jakość komunikacji między człowiekiem a maszyną.

Ogólnie, można wyróżnić (nie tylko w bibliotece CP/M) trzy podstawowe grupy pakietów: zarządzania bazą danych, przetwarzania tekstów oraz operowania tzw. elektronicznym arkuszem (ang. spreadsheet).

Pakiety zarządzania bazą danych są szeroko stosowane w dużych komputerach. Wersje mikrokomputerowe cechują

Wyniki testowania kompilatorów wybranych języków pod systemem CP/M (wg [2])

Program A	BASIC-80 (MICROSOFT)	FORTRAN-80 (MICROSOFT)	PASCAL/Z (ITHACA INTERSYSTEM)	PL/I-80 (DIGITAL RESEARCH)	COBOL-80 (MICROSOFT)
Czas kompilacji (s)	17,8	16,1	37,8	39,4	54,0
Czas łączenia (s)	158,0	62,2	58,7+, 48,5	70,8	87,5
Czas wykonania (s)	115,8	3,9	26,3	4,2	1275,7
Plik źródłowy (wielokrotność 128 bajtów)	4	4	5	5	14
Plik typu .COM (wielokrotność 128 bajtów)	98	60	28	61	118
Program B					
Czas kompilacji (s)	16,7	17,4	33,7	38,2	50,2
Czas łączenia (s)	158,2	69,2	74,0+,52,4	85,4	88,5
Czas wykonania (s)	20,8	16,6	104,5	28,9	325,7
Plik źródłowy (wielokrotność 128 bajtów)	2	3	3	3	9
Plik typu .COM (wielokrotność 128 bajtów)	82	57	40	68	122

się dużą łatwością obsługi, oparciem na dialogu i dużej ilości informacji pomocniczych i wyjaśnień. W bibliotece CP/M występują następujące pakiety: CONDOR — relacyjna baza danych ze zwiększoną możliwością tworzenia formatek ekranowych, HDBS — hierarchiczna baza danych (wg CODASYL), DATENBANK — relacyjna baza danych (wg Codda) oraz najbardziej znana — dBASE II — relacyjna baza danych o następujących parametrach: maks. 65 535 rekordów w pliku, maks. 1000 znaków w rekordzie, maks. 254 znaków w polu i do 100 znaków klucza indeksowego.

Arkusze elektroniczne są nowym zjawiskiem w informatyce. Pierwszym tego rodzaju programem był VISICALC dla APPLE II. Obecnie istnieje kilkadziesiąt pakietów, o różnych możliwościach i cenie, ale z zachowaniem podstawowej zasady — umożliwiającej przeprowadzanie skomplikowanych operacji przetwarzania danych liczbowych i napisów na rozległym arkuszu (formularzu) odwzorowanym w pamięci komputera. Fragment arkusza wyświetlany na ekranie można łatwo przesuwać, uzyskując dostęp do dowolnego elementu danych. Możliwość tworzenia algorytmów powiązań między polami arkusza, dokonywania operacji arytmetycznych i logicznych — powoduje, że pakiety te nadają się szczególnie do przeprowadzania kalkulacji, planowania (arkusze planistyczne), obliczeń inżynierskich, tworzenia zestawień tekstowo-cyfrowych itp. Obsługa pakietów jest prosta i opiera się na tzw. menu — zestawie funkcji, które wybiera się przesuwając kursor za pomocą klawiszy. Dodatkowo, w większości pakietów można w dowolnej chwili otrzymać szczegółowe informacje o parametrach i funkcjonowaniu pakietu. W bibliotece CP/M istnieją następujące arkusze elektroniczne: SUPERCALC, CALCSTAR, KNOWLEDGEMAN, MULTIPLAN, TABPLAN-CSK.

Pakiety przetwarzania tekstów pozwalają na redagowanie tekstów na całym ekranie (standardowy edytor ED.COM jest edytorem liniowym), przeszukiwanie plików, łączenie bloków tekstów, przenoszenie, przesuwanie, ustalanie postaci (formatu) wydruków i wykonywanie wielu innych funkcji związanych z tworzeniem plików danych tekstowych. Do najbardziej znanych pakietów należą: WORDMASTER, VEDIT i WORD STAR. Z tym ostatnim mogą współpracować: MAILMERGE, ułatwiający pisanie typowych listów, drukowanie adresów itp., oraz FOOTNOTE, ułatwiający wprowadzanie przypisów na poszczególnych stronach tekstu.

Do pakietów użytkowych można zaliczyć także: MILESTONE, który wspomaga zarządzanie przedsięwzięciami projektowymi (wykorzystując metodę PERT), STATPAK — profesjonalny pakiet statystyczny o rozbudowanych funkcjach z zakresu rachunku prawdopodobieństwa i statystyki, oraz muMATH/muSIMP — zapewniający rozwiązywanie problemów matematycznych w dialogu z komputerem.

INNE PROGRAMY

Pozostałymi godnymi odnotowania pozycjami biblioteki są programy, które zmieniają sposób komunikowania się użytkownika z systemem CP/M (na poziomie programu CCP [1]). SMARTSCREEN eliminuje jedną z dokuczliwych wad systemu — liniowe wprowadzanie i wyprowadzanie danych, gdyż umożliwia pracę na całym ekranie. SMARTKEY i SMARTPRINT pozwalają na dopasowanie niestandardowej klawiatury do standardu CP/M oraz przyporządkowanie jednemu klawiszowi całego ciągu znaków.

Kuracją odmładzającą dla CP/M nazwano wprowadzenie programu MICROSHELL, który zastępuje CCP i tworzy analizator poleceń o właściwościach powłoki (ang. shell) systemu UNIX. W rezultacie uzyskuje się szereg dotychczas niedostępnych możliwości, jak np. przesyłanie danych do i z dowolnego urządzenia zewnętrznego lub przekazywanie parametrów z jednego programu do drugiego metodą tłoczenia (ang. pipes).

Oddzielną grupę stanowią programy graficzne, np. CP/M GRAPHICS firmy DIGITAL RESEARCH ukierunkowany na standard ANSI-VDL. Zawiera on wiele procedur ułatwiających tworzenie obrazów graficznych oraz dopasowanie pakietu do specyfiki konkretnego sprzętu.

* * *

Przedstawione programy i pakiety są oczywiście tylko fragmentem biblioteki CP/M. Jest jednak pewne, że użytkownicy mikrokomputerów ELWRO 513, IMP 85 lub LIDIA chętnie widzieliby podobne oprogramowanie w swojej bibliotece. Warto więc pomyśleć nad adaptacją takich programów do warunków polskich, a nie pisać ich od nowa.

LITERATURA

[1] Kern C. O.: Five C Compilers for CP/M-80. BYTE, vol. 8, No. 8, p. 110 (1983)

[2] Loose A.: Compiler-Sprachen unter CP/M. MC — Die Mikrocomputer Zeitschrift, nr 10, s. 49, 1983

[3] Sobczyk M.: Dyskowy system operacyjny CP/M. Cz. 1—3. INFORMATYKA, nr 2, 3, 4/1983.

[1]) CCP — ang. communication command processing, przetwarzanie poleceń operatorskich

RYSZARD RYBUS
Warszawa

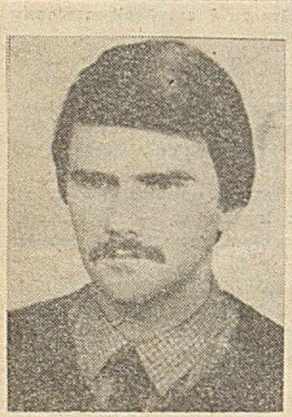
Oprogramowanie systemu mikroprocesorowego bez pamięci masowej

Systemy mikroprocesorowe bez pamięci masowej są wyposażone w proste oprogramowanie, które najczęściej składa się z makroasemblera i programu uzdatniającego (ang. monitor-debugger). W artykule opisano koncepcję oprogramowania, której celem jest zachowanie udogodnień tradycyjnego systemu przetwarzania (tzn. możliwości redagowania, tłumaczenia i uruchamiania programów użytkowych) przy uwzględnieniu braku dostępu do pamięci masowej [1]. W oparciu o opisaną koncepcję, autor zaprojektował i zrealizował system SPM (System Programowania MC 6800) przeznaczony dla mikroprocesora MOTOROLA 6800.

TWORZENIE PROGRAMU ŹRÓDŁOWEGO

Podstawowym warunkiem skutecznej pracy nad tworzeniem programu jest łatwy dostęp do jego postaci źródłowej — w celu poprawy błędów składniowych i logicznych. W typowych systemach program jest przechowywany w pamięci masowej, np. dyskowej czy kasetowej, co zapewnia możliwość szybkiego oraz wygodnego dostępu. Sytuacja jest trudniejsza w systemach dysponujących tylko taśmą papierową. Na ogół taśmy papierowej nie wykorzystuje się jako podstawowego nośnika dla programu źródłowego, gdyż jest to nośnik zawodny i kłopotliwy w użyciu. Dlatego w takich systemach łatwo dostępny może być tylko program przechowywany w pamięci RAM. Program źródłowy stale przebywający w tej pamięci wymaga odpowiednio dużego obszaru (średnio 3 K bajty na 100 instrukcji).

W celu zminimalizowania zajętości pamięci, program źródłowy przechowuje się w postaci pośredniej. Należy przy tym pamiętać, że:
— kodowanie instrukcji w postaci pośredniej powinno być maksymalnie efektywne
— musi istnieć łatwe przejście od postaci pośredniej do postaci źródłowej (co zapewnia faktyczną realizację dostępu do postaci źródłowej).
Postulat efektywnego kodowania instrukcji wymaga, aby język pośredni był podobny do języka maszynowego, co umożliwia również zredukowanie części systemu dokonującej translacji programu na postać wynikową. W projekcie języka pośredniego dużą rolę odgrywają cechy źródłowego języka asemblerowego. Z uwagi na postulat oszczędnego wykorzystania pamięci, należy wprowadzić do języka asemblera ograniczenia w stosowaniu etykiet i komentarzy. Ograniczenia takie mogą przynieść istotne zmniejszenie pamięci niezbędnej do reprezentacji postaci źródłowej i tablicy symboli. W systemie SPM przyjęto, że w programie można użyć do stu etykiet o postaci H0—H99. Komentarz jest zapamiętany bezpośrednio za rekordem reprezentującym instrukcję w postaci pośredniej, ale w dowolnej chwili wszystkie komentarze mogą być usunięte z programu odpowiednim poleceniem systemu.



Mgr inż. RYSZARD RYBUS ukończył studia w marcu 1983 w Instytucie Informatyki Politechniki Warszawskiej. W ramach pracy dyplomowej zaprojektował i zrealizował system programowania dla mikroprocesora MOTOROLA 6800.

Przykład budowy rekordu instrukcji w postaci pośredniej w systemie SPM przedstawiono w tabeli 1. Zajętość pamięci przez linię programu w postaci źródłowej, pośredniej i wynikowej, w zależności od użytego trybu adresowania, zilustrowano w tabeli 2. Wynika z niej, że zastosowanie postaci pośredniej — zamiast postaci źródłowej — umożliwia uzyskanie ponad 50% oszczędności w wykorzystaniu pamięci.

Tabela 1. Przykład budowy rekordu instrukcji H23 NEG FF00 w postaci pośredniej

Numer bajtu rekordu	Znaczenie	Wartość bajtu dla instrukcji H23 NEG FF00 (zapis szesnastkowy)
1	Bajt opisu rekordu	26
2	Bajt opisu instrukcji	D8
3	Kod operacji dla danej instrukcji	70
4	Argument instrukcji (od 0 do 2 bajtów)	FF
5		00
6	Etykieta zdefiniowana w rekordzie	23

Bajt opisu rekordu		Bajt opisu instrukcji	
Bity	Znaczenie	Bity	Znaczenie
0—3	Długość rekordu	0	Wskaźnik argumentu w postaci etykiety
4	Wskaźnik komentarza	1—2	Typ stałej użytej jako argument
5	Wskaźnik zdefiniowania etykiety	3—5	Tryb adresowania
6	Rekord komentarza	6—7	Długość kodu wynikowego
7	Rekord dyrektywy		

Każda wprowadzona linia programu jest bezpośrednio tłumaczona z postaci źródłowej na postać pośrednią. Na tym etapie translacji wykrywane są błędy syntaktyczne, których wystąpienie powoduje odrzucenie niepoprawnej linii wejściowej. Bezpośrednia sygnalizacja błędów syntaktycznych zdecydowanie przyspiesza proces powstawania programu. Wynika to z faktu, że większość błędów syntaktycznych powstaje przy wprowadzaniu programu, a ich przyczyną jest nieuwaga użytkownika.

Tabela 2. Porównanie zajętości pamięci w bajtach przez linię programu w różnych postaciach (bez komentarza)

Tryb adresowania MC 6800	Postać wynikowa	Postać pośrednia	Postać źródłowa*)
Rejestrowy	1	3—4	4—10
Prosty (ang. immediate)	2—3	4—6	10—15
Strony zerowej	2	4—6	9—13
Indeksowy	2	4—5	11—15
Bezwzględny	3	5—6	9—13
Względny	2	4—6	9—13

*) Przyjęto argument 4-znakowy

System SPM realizuje następujące funkcje przeznaczone do redagowania tekstu programu:

- wstawianie linii programu źródłowego do pamięci (usuwanie ich z pamięci)
- wyprowadzanie linii programu źródłowego
- usuwanie komentarzy z programu źródłowego
- określanie położenia programu źródłowego w pamięci
- wyprowadzanie informacji o obszarze zajmowanym przez program.

PROCES TŁUMACZENIA

Translacji do postaci wynikowej podlega program źródłowy w postaci pośredniej. Powstający program wynikowy (nieprzesuwany) jest wstawiany bezpośrednio do pamięci. Wartość licznika rozkazów, według którego dokonywana jest translacja, określa dyrektywa ORG języka asemblera. Jeśli dyrektywa ORG nie występuje, to licznik rozkazów otrzymuje wartość równą pierwszemu adresowi za programem źródłowym. W ten sposób program źródłowy i wynikowy mogą zajmować spójny obszar pamięci. W systemie SPM przewidziano możliwość załadowania programu wynikowego pod inne adresy niż te, według których został on przetłumaczony. Ma to zastosowanie w przypadku, gdy docelowy obszar pamięci przeznaczony dla programu jest aktualnie niedostępny.

Translacja programu jest wykonywana w dwóch przebiegach, co umożliwia umieszczenie w raporcie pełnej informacji o programie wynikowym. Na podkreślenie zasługuje duża szybkość procesu translacji — jej przyczyny są następujące:

— główną część tłumaczenia instrukcji na kod maszynowy wykonuje się w fazie redagowania tekstu programu
— proces tłumaczenia angażuje wyłącznie pamięć (wyłączając wyprowadzenie raportu).

Asembler tworzy wyłącznie nieprzesuwany kod wynikowy, a do relokacji programu używa się dyrektywy ORG. Proces relokacji, polegający na powtórnej translacji po wstawieniu dyrektywy ORG do programu, jest bardzo szybki. Stanowi to konsekwencję łatwego dostępu do postaci źródłowej i szybkiej procedury tłumaczenia programu.

Do łączenia programów wykorzystuje się raport wyprowadzany podczas translacji (adres początku i końca kodu wynikowego oraz tabelę etykiet) i dyrektywę EQU służącą do definiowania wartości etykiety.

URUCHAMIANIE PROGRAMU

Część poleceń systemu SPM, realizujących funkcje programu uzdatniającego, jest przeznaczona do uruchamiania programu. Program uzdatniający umożliwia:

- wyprowadzenie lub modyfikowanie zawartości komórek pamięci i rejestrów (w postaci szesnastkowej)
- wyprowadzenie lub modyfikowanie programu w postaci symbolicznej
- przesuwanie zawartości wskazanego bloku pamięci
- inicjowanie programu użytkowego od wskazanego miejsca
- zatrzymywanie programu użytkowego w miejscach określonych z góry
- krokowe wykonywanie programu (rozkaz po rozkazie)
- śledzenie wykonywania wskazanej liczby rozkazów
- sygnalizację niesekwencyjnej zmiany sterowania w programie użytkowym (np. rozkazów skoku).

W wymianie sterowania między programem użytkowym a systemem stosowany jest mechanizm pułapek programowych [2].

W systemie SPM istnieje możliwość korzystania w czasie uruchamiania programu z symbolicznych elementów języka asemblera. W tej klasie oprogramowania, w której mieści się opisywany system, jest to rzadko wprowadzane udogodnienie. Program uzdatniający korzysta z tabeli etykiet utworzonej w fazie translacji. Wynika z tego możliwość używania wyrażeń symbolicznych jako parametrów odpowiednich poleceń. Składnikami wyrażeń mogą być zdefiniowane etykiety oraz stałe dziesiętne i szesnastkowe. Uruchamianie symboliczne uwalnia użytkownika od uciążliwego stosowania fizycznych adresów pamięci. SPM ma polecenia służące do obsługi tabeli etykiet; możliwe jest np. definiowanie etykiety. Dodatkowo użytkownik może wyprowadzać lub modyfikować zawartość pamięci w postaci instrukcji.

* * *

Przedstawiona koncepcja umożliwia realizację oprogramowania przeznaczonego do tworzenia niewielkich programów, o wielkości kodu wynikowego do 2 K bajtów. W przypadku większych programów, do przetwarzania postaci źródłowej należy stosować asembler skrośny, a do uruchamiania i testowania można wykorzystać program uzdatniający wchodzący w skład systemu SPM.

Oprogramowanie podobne do systemu SPM najwygodniej jest umieścić w pamięci ROM. Wymaga to zmniejszenia, na ile to jest możliwe, wielkości pamięci zajmowanej przez oprogramowanie systemowe. Wiąże się z tym konieczność odpowiedniej organizacji struktury danych, która ma wpływ na wielkość zajmowanej pamięci i na szybkość wyszukiwania informacji. Warunek efektywnego wyszukiwania ma szczególnie duże znaczenie, gdyż szybkość reakcji konwersacyjnego systemu programowania przesądza o jego jakości.

```
*AL ; TRANSLACJA PROGRAMU
LISTING BY SPM ASSEMBLER
LOCN B1 B2 B3 REKORD
...  ; KOD PROGRAMU
BYTES ... ERRORS ...
UNDEFINED SYMBOLS ...
SYMBOLS ...
*DM H1 ; USTAWIENIE ADRESU TABLICY 1
...
*DL H50:3804;DEFINIOWANIE ETYKIETY
*PM 3800 3 ;ZAWARTOSC TABLICY 1
...
*PM H50,3 ;ZAWARTOSC TABLICY 2
...
*DS ;USTAWIENIE PARAMETROW W REJESTRACH
...
*SB H5+4 ;USTAWIENIE PULAPKI STALEJ
...
*GO H10:H4 ;START PROGRAMU
BREAK AT ...
... ;STAN REJESTROW
*TR ;PRACA KROKOWA
... ;STAN REJESTROW
*TR 2 ;SLEDZENIE
... ;STAN REJESTROW
*SJ ;WSTRZYMANIE PROGRAMU PO SKOKU
BREAK AT
... ;STAN REJESTROW
*GO ;WYKONANIE PROGRAMU DO KONCA
BREAK AT...
... ;STAN REJESTROW
```

Fragment procesu tworzenia programu w systemie SPM (translacja i uruchamianie)

Proces tworzenia programu w systemie SPM, którego przykładowy fragment przedstawiono obok, jest wygodny i efektywny. Stała obecność programu źródłowego w pamięci RAM umożliwia szybkie wprowadzanie poprawek i modyfikacji. Zastosowanie edytora, dokonującego częściowej translacji, umożliwiło skrócenie czasu traconego zwykle na poprawę błędów syntaktycznych. Proces usuwania błędów logicznych również nie jest zbyt kłopotliwy, z uwagi na bogaty zestaw funkcji uruchomieniowych i możliwość uruchamiania symbolicznego.

LITERATURA

[1] Fernstrom C., Kruzela I., Svensson B.: Asmedit — A New Philosophy for Program Development. Proc. 4th Euromiero Symposium, Munich, 1978, North-Holland, Amsterdam, 1978

[2] Gondzio M.: Projektowanie, realizacja i zastosowanie pułapek programowych w monitorach-debuggerach dla mikroprocesorów 8-bitowych. Raporty Badawcze Instytutu Informatyki PW, nr 42, 1982.

WOJCIECH TROJNIAR
Zakład Systemów Automatyki Kompleksowej
Polska Akademia Nauk
Gliwice

7/84

FORTH – język i system programowania (2)

W pierwszej części artykułu przedstawiono model FORTHA jako pewnej maszyny wirtualnej, którą nazwano maszyną FORTH. Opisano również operacje arytmetyczno-logiczne i operacje na stosie zdefiniowane w języku FORTH. Poniżej zostaną opisane dalsze elementy języka, a w szczególności — operacje na pamięci, instrukcje strukturalne i operacje wejścia-wyjścia. Ponadto zostaną przedstawione elementy systemu FORTH oraz opis przygotowania programów za pomocą tzw. FORTH-edytora.

OPERACJE NA PAMIĘCI

Zmienne w języku FORTH są definiowane przy użyciu słowa VARIABLE. Parametrami tego słowa są — wartość początkowa zmiennej i jej nazwa. Po zdefiniowaniu zmiennej, słownik zostanie rozszerzony o odpowiednie słowo. Przykładowo — napis:

```
0 VARIABLE RESULT
```

oznacza utworzenie słowa o nazwie RESULT i wartości początkowej 0. Po wywołaniu słowa RESULT (z klawiatury terminala lub z programu) na szczyt stosu zostanie przesłany adres, pod którym znajduje się wartość tej zmiennej. Standardowo, każdej zmiennej odpowiadają 2 bajty do przechowania jej wartości.

Wymiana informacji między stosem a pamięcią odbywa się przy użyciu następujących operacji (p. część 1 artykułu):

- załadowanie 16-bitowej danej **n** z komórki pamięci o adresie **addr** na szczyt stosu[1]):

```
@ ( addr → n )
```

- zapamiętanie 16-bitowej danej **n** pod adresem **addr**:

```
! ( n addr → )
```

Dla danych o długości 1 bajt i podwójnej długości (4 bajty) zdefiniowano analogiczne operacje: **C@**, **C!** i **2@**, **2!**. FORTH dopuszcza operacje na pamięci, która nie została jawnie zadeklarowana. Przed wykonaniem operacji na komórce pamięci, użytkownik przesyła jej adres na szczyt stosu. Operacja jest wykonywana bez żadnego sprawdzenia legalności dostępu do danego obszaru pamięci. Przykładowo — przesłanie danych między zmiennymi **V1** i **V2** nastąpi po wykonaniu programu:

```
V1 @ V2 !
```

Poniższe operacje umożliwiają wypełnianie obszarów pamięci wzorcem oraz przesyłanie danych między obszarami pamięci:

- wypełnienie **u** bajtów pamięci wzorcem **b** począwszy od adresu **addr**:

```
FILL ( addr u b → )
```

- wypełnienie **u** bajtów pamięci zerami, począwszy od adresu **addr**:

```
ERASE ( addr u → )
```

- wypełnienie **u** bajtów pamięci spacjami począwszy od adresu **addr**:

```
BLANKS ( addr u → )
```

- przesłanie **u** bajtów z obszaru pamięci zaczynającego się od adresu **addr1** do obszaru pamięci zaczynającego się od adresu **addr2**:

```
CMOVE ( addr1 addr2 u → )
```

[1]) W nawiasach okrągłych, jako komentarz, podano zmiany na szczycie stosu wywołane wykonaniem operacji

INSTRUKCJE STRUKTURALNE

Forth jest językiem strukturalnym, tzn. jego słownik jest wyposażony w repertuar słów umożliwiających budowanie struktur programowych. Zagnieżdżanie struktur jest ograniczone pojemnością stosów.

Nieskończoną pętlę programową tworzy para słów:

```
BEGIN ... AGAIN
```

Kod programu zawarty w tej pętli jest wykonywany cyklicznie. Para słów:

```
BEGIN ... UNTIL
```

tworzy instrukcję iteracyjną. Kod zawarty między tymi słowami wykonuje się powtarzalnie — aż do chwili, gdy przed wykonaniem instrukcji UNTIL na szczycie stosu pojawi się wartość logiczna **true**.

W instrukcji iteracyjnej:

```
BEGIN ... WHILE ... REPEAT
```

kod zawarty między słowami BEGIN i REPEAT wykonuje się powtarzalnie aż do chwili, gdy w czasie wykonania programu słowa WHILE wartość logiczna na szczycie stosu wynosi **false**. Następnie program jest wykonywany od słowa, które w tekście źródłowym znajduje się bezpośrednio po słowie REPEAT.

Do wykonania segmentu programu określoną liczbę razy używa się instrukcji:

```
DO ... LOOP lub DO ... +LOOP
```

Parametrami wejściowymi tych instrukcji są wartości indeksów: końcowego, powiększonego o 1 oraz początkowego, umieszczone na szczycie stosu. Program zawarty w tej konstrukcji wykonuje się dotąd, aż indeks pętli osiągnie lub przewyższy wartość indeksu końcowego. Wartość indeksu jest zwiększana o 1 przez program słowa LOOP lub — o wartość znajdującą się na szczycie stosu przez program słowa +LOOP. Program zawarty wewnątrz pętli może odczytać aktualną wartość indeksu pętli przez wywołanie słowa I. Wykonanie programu słowa LEAVE wewnątrz pętli spowoduje zmianę wartości aktualnego indeksu na stosie powrotnym (gdzie przechowywane są indeksy pętli) do wartości końcowej. Po wykonaniu programu słowa LOOP lub +LOOP nastąpi opuszczenie pętli.

Do warunkowego wykonania programu służą instrukcje:

```
IF ... ENDIF oraz IF ... ELSE ... ENDIF
```

Po napotkaniu instrukcji IF, o kontynuacji programu decyduje wartość logiczna pobierana ze szczytu stosu. Jeżeli wartość ta jest równa **true**, to wykonuje się program zawarty między słowami

```
IF ... ENDIF lub IF ... ELSE
```

Instrukcje strukturalne nie mogą być wykonywane interpretacyjnie z klawiatury terminala lub z dysku. Mogą one być wywoływane jedynie z programu.

Z instrukcjami strukturalnymi związane są operacje testowania, które służą do obliczania wartości logicznych wyrażeń na podstawie danych znajdujących się na szczycie stosu. Wartości te, pozostawiane na stosie, są parametrami dla instrukcji strukturalnych. Wyróżniono następujące operacje testowania (**f** jest równe **true**, gdy odpowiednie relacje są spełnione):

- badanie równości:

```
= ( n1 n2 → f )
```

7/84*

- badanie mniejszości (n1<n2):

< (n1 n2 → f)

- badanie większości (n1>n2):

> (n1 n2 → f)

- badanie równości z zerem:

0= (n → f)

- badanie ujemności:

0< (n → f)

- badnie dodatności:

0> (n → f)

Omówione instrukcje ilustruje przykładowy program słowa PRINT-NATURAL-NUMBERS, drukujący w kolejnych liniach liczby naturalne, począwszy od liczby 1.

```
: PRINT-NATURAL-NUMBERS
        DUP 0= IF
        1+ 1 DO I CR . LOPP
           ENDIF ;
```

Program słowa **CR** wysyła na terminal ciąg znaków **CR** i **LF**, zaś program słowa . — wypisuje na terminalu liczbę ze szczytu stosu.

OPERACJE WEJŚCIA-WYJŚCIA

Standardowe operacje wejścia-wyjścia umożliwiają współpracę z terminalem lub dyskiem. Komunikacja z terminalem polega na przesyłaniu pojedynczych znaków lub napisów do lub z terminala. Komunikacja z dyskiem jest wykonywana według przyjętych reguł wynikających z organizacji logicznej dysku.

Komunikacja z terminalem

Program wykonywany w systemie FORTH może odczytywać znaki z klawiatury terminala przy użyciu słowa KEY. Po wywołaniu programu tego słowa, kod znaku odpowiadającego przyciśniętemu klawiszowi jest przesyłany na szczyt stosu. Słowem ?TERMINAL sprawdza się, czy klawisz jest wciśnięty. Jeżeli tak, to na stos przesyłana jest wartość **true**, w przeciwnym przypadku — **false**.

Użytkownik może sam zorganizować współpracę z terminalem lub korzystać z programów dostarczonych wraz z systemem. W systemie przewidziano bufor do komunikacji z terminalem i pewną liczbę procedur operujących na tym buforze. Posługując się słowem QUERY, można gromadzić tekst w buforze wejściowym terminala aż do napotkania znaku **CR** lub do przepełnienia bufora. Fragmenty tekstu z bufora są przesyłane przy użyciu słowa WORD do tzw. bufora WORD, gdzie podlegają interpretacji przez interpreter zewnętrzny lub program użytkownika (p. część 1 artykułu).

W szczególności, tekstem w buforze może być liczba. Użycie słowa NUMBER powoduje zamianę tekstu znajdującego się w buforze WORD na liczbę podwójną, która może zawierać kropkę dziesiętną. Pozycja kropki, przechowywana przez specjalną zmienną o nazwie DPL, nie wpływa na wartość liczby przesłanej na szczyt stosu. Jeżeli liczba nie zawiera kropki dziesiętnej, to wartość zmiennej DPL jest równa —1. Kolejna pozycja liczby występująca po kropce dziesiętnej zmniejsza wartość zmiennej DPL o 1. Łatwo więc stwierdzić, że w celu obliczenia cechy liczby, wystarczy do jej liczby pozycji dodać wartość zmiennej DPL.

Interpreter zewnętrzny zamienia liczbę z kropką dziesiętną na liczbę podwójnej długości, a bez kropki dziesiętnej — na liczbę pojedynczej długości (16 bitów). Konwersja liczby z kodu ASCII na postać dwójkową jest dokonywana według podstawy numerycznej określonej wartością zmiennej BASE. Użytkownik może zmienić podstawę przy użyciu słowa HEX lub DECIMAL albo przez zmianę wartości zmiennej BASE.

Przy użyciu słowa EMIT wysyła się znak ze szczytu stosu na ekran terminala. Do przesyłania tekstów na monitor używa się następujących słów:

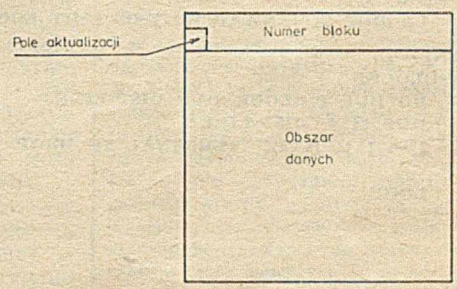
CR — przesłanie znaków **CR** i **LF**
SPACE — przesłanie spacji w kodzie ASCII
SPACES (n→) — przesłanie **n** znaków spacji w kodzie ASCII
TYPE (addr count→) — przesłanie **count** znaków spacji w kodzie ASCII
." — użyte w postaci **."cccc"** powoduje wyprowadzenie na ekran tekstu **cccc** aż do napotkania znaku " lub zapamiętanie tego tekstu w pamięci (można go następnie wyprowadzić, używając tego słowa w programie).

Liczby są wyprowadzane według podstawy numerycznej, określonej wartością zmiennej BASE. Konwersja liczby jest dokonywana w buforze zwanym PAD. Wyprowadzenie liczby na ekran terminala następuje przy użyciu następujących słów:

. (n→) — przesłanie liczby pojedynczej długości ze znakiem
U. (u→) — przesłanie liczby pojedynczej długości bez znaku
D. (d→) — przesłanie liczby podwójnej długości ze znakiem
.R (n1 n2→) — przesłanie liczby pojedynczej długości ze znakiem **n1**; liczba jest przesunięta na prawo w polu o długości **n2**
D.R (d n→) — przesłanie liczby podwójnej długości ze znakiem; liczba jest przesunięta na prawo w polu o długości **n**
? (addr→) — przesłanie liczby znajdującej się w pamięci pod adresem **addr.**

Komunikacja z dyskiem

Organizacja pamięci dyskowej jest oparta na podstawowej jednostce logicznej zwanej blokiem. Pamięć jest zorganizowana jako jednowymiarowy wektor bloków numerowanych od zera. Bloki są grupowane w większe jednostki zwane kadrami (ang. screen)[2]. Kadr składa się z 1024 znaków podzielonych na szesnaście 64-znakowych linii. Kadr zawiera całkowitą liczbę bloków, zależnie od implementacji. Zwykle blok jest równoważny sektorowi dyskowemu lub równy kadrowi.



Rys. 1. Struktura bufora dyskowego

Komunikacja z dyskiem odbywa się za pośrednictwem buforów dyskowych. Każdy bufor zajmuje obszar pamięci równy jednemu blokowi. Liczba buforów jest stała, zadeklarowana w czasie powstawania systemu. Bufor dyskowy (rys. 1) składa się z trzech pól. Pole numeru bloku zawiera aktualny numer tego bloku, którego treść znajduje się w polu danych bufora. Zawartość pola aktualizacji wskazuje, czy zasygnalizowano zmianę wartości bufora. Użytkownik może komunikować się z dyskiem przez wywołanie programu słowa BLOCK, który pełni funkcje pamięci wirtualnej:

BLOCK (n→addr)

Na podstawie numeru bloku **n**, program podaje adres pola danych bufora. Najpierw sprawdza się pola numerów bloków. Jeżeli nie znaleziono bloku o zadanym numerze, to sprawdza się, czy pole aktualizacji najwcześniej odczytanego bloku dyskowego zawiera wartość **true**. Jeżeli tak, to zawartość bufora jest przepisywana do odpowiedniego bloku na dysk. W ten sposób zwalnia się miejsce do odczytu zadanego bloku. Następnie do tego bufora odczytuje się blok z dysku. Pole numeru bloku jest aktualizowane, a na pole aktualizacji wpisuje się wartość **false**.

---

[2]) Mimo że screen znaczy dosłownie ekran, rozsądniej jest dobrać taki odpowiednik polski, który nie tylko odpowiada treści pojęcia, ale jednocześnie zachowuje zgodność semantyczną z pozostałymi wyrazami zdania (przyp. red.)

Znaczenie pozostałych słów przeznaczonych do komunikacji z dyskiem opisano poniżej:

UPDATE — wpisanie wartości **true** na pole aktualizacji ostatnio używanego bufora

FLUSH — przepisanie na dysk wszystkich buforów, których pola aktualizacji zawierają wartość **true**

DR0 — wybór mechanizmu dyskowego o numerze **0** (słowo BLOCK służy do komunikacji z uprzednio wybranym mechanizmem dyskowym)

DR1 — wybór mechanizmu dyskowego o numerze **1**

LIST (**n** →) — przesłanie na terminal tekstu kadru dyskowego o numerze **n**

LOAD (**n** →) — interpretacja tekstu z dysku od kadru o numerze **n**

→ — kontynuacja interpretacji tekstu z kolejnego kadru

;S — zakończenie interpretacji tekstu z dysku.

Poniżej zamieszczono program wydrukowany przy użyciu słowa LIST, który przepisuje zawartość bloków od numeru **n1** do numeru **n2-1** — do bloków zapoczynających się od numeru **n3**. Zawartość każdego bloku jest wypisywana na ekranie terminala. Wykonanie programu może być przerwane przez naciśnięcie klawisza z klawiatury terminala.

```
SRC # 150
  0 ( KOPIOWANIE BLOKOW DYSKOWYCH )
  1 ( BLOKI OD NUMERU N1 DO NUMERU N2-1
                                  SA KOPIOWANE )
  2 ( NA DYSK POCZAWSZY OD NUMERU BLOKU N3.
                              KOPIOWANA ZAWARTOSC )
  3 ( JEST WYSWIETLANA NA EKRANIE TERMINALA
                             W 64 ZNAKOWYCH LINIACH )
  4
  5 ( WYPISANIE NA EKRAN 1 LINII TEKSTU )
  6 ( N → )
  7 : DISP-LINE
  8         CR DUP 64 + SWAP DO I C@ EMIT LOOP ;
  9 ( N3 N2 N1 → )
 10 : COPY-BLOCKS
 11         DO DUP I BLOCK DUP DISP-LINE
 12         DUP 64 + DISP-LINE
 13         2 — ! UPDATE ?TERMINAL IF DROP LEAVE
                                          ENDIF 1+
 14         LOOP ;
 15 ;S
OK
```

SŁOWNIKI

Słowa tworzą listę połączoną lub strukturę drzewiastą. Każdą z gałęzi drzewa nazywamy s ł o w n i k i e m. Pniem drzewa jest słownik o nazwie FORTH. W określonej chwili dostępne są dwie listy połączone słów. Pierwsze słowo listy jest liściem, a ostatnie — korzeniem drzewa. Adresy słowników aktualnie dostępnych są określone wartościami zmiennych CURRENT i CONTEXT. Nowe słowo jest dołączone do słownika CURRENT. W czasie interpretacji tekstu interpreter zewnętrzny najpierw przeszukuje słownik CONTEXT, a jeżeli nie znajdzie nazwy szukanego słowa, to przechodzi do przeszukiwania słownika CURRENT.

Nowy słownik jest tworzony przy użyciu słowa VOCABULARY, np.:

```
VOCABULARY EDITOR
```

powoduje utworzenie słownika o nazwie EDITOR. Jest on dołączony do słownika wskazanego przez zmienną CURRENT. Po użyciu nazwy słownika (np. EDITOR), następuje wpisanie adresu tego słownika do zmiennej CONTEXT. Program słowa DEFINITIONS przepisuje wartość zmiennej CONTEXT do zmiennej CURRENT. Wywołanie:

```
EDITOR DEFINITIONS
```

umożliwia dołączenie kolejnych słów do słownika EDITOR. Zdefiniowanie słownika o nazwie DEBUGGER w słowie FORTH ma następującą postać:

```
FORTH DEFINITIONS VOCABULARY DEBUGGER
```

Po zdefiniowaniu słów w tym słowniku i wywołaniu słowa EDITOR, można korzystać z repertuaru słów zdefiniowanych w słownikach FORTH, EDITOR i DEBUGGER.

Poniżej podano niektóre częściej używane słowa do operacji związanych ze słownikiem:

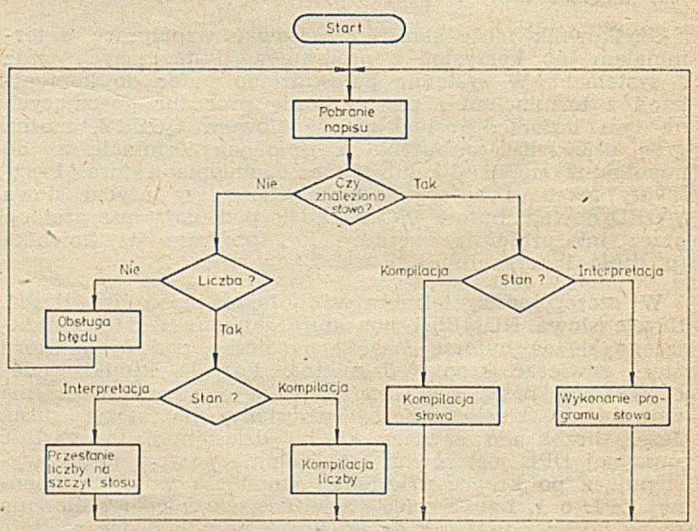
—FIND (→ **pfa b tf**) lub (→ **ff**) — pobranie tekstu z bufora terminala lub dysku do bufora WORD i sprawdzenie, czy takie słowo istnieje w słowniku CONTEXT lub CURRENT; jeżeli znaleziono słowo, to na stos są przesyłane parametry słowa **(pfa,b)** i wartość **true**, w przeciwnym przypadku na stos jest przesłana wartość **false**

FORGET ccc — usunięcie ze słownika słowa o nazwie **ccc** oraz wszystkich słów po nim zdefiniowanych

' (→ **addr**) — przesłanie na szczyt stosu adresu pola parametrów słowa **ccc**.

PRZYGOTOWANIE PROGRAMU ŹRÓDŁOWEGO

Program źródłowy przygotowuje się w postaci kadrów dyskowych. Do tego celu służy tzw. FORTH-edytor, który umożliwia redagowanie tekstu w ramach jednego kadru. Do funkcji edytora należy: wypisywanie, usuwanie i wstawianie linii; poszukiwanie, wstawianie i usuwanie wzorców oraz przesuwanie kursora po ekranie. Jeżeli interpretacja przygotowanego programu ma być kontynuowana na następnym kadrze, to bieżący kadr należy zakończyć słowem →. Ostatni kadr programu kończy się słowem **;S**. Przygotowany tekst programu interpretuje się przy użyciu słowa LOAD. Schemat blokowy interpretera zewnętrznego przedstawiono na rysunku 2 (z pominięciem gromadzenia znaków w buforze).



Rys. 2. Schemat blokowy interpretera zewnętrznego

* * *

W trzeciej części artykułu zostanie omówiona technika implementacji FORTHA oraz zamieszczony przykład dłuższego programu w tym języku.

Opisywaliśmy już w INFORMATYCE system operacyjny CP/M [2]. Obecnie możemy stwierdzić, że jest to system, który najbardziej rozpowszechnił się w kraju. Dlatego też nie powinno być dużych kłopotów ze zdobyciem zawierającej go dyskietki. To jednak dopiero początek. Co zrobić dalej napisano w poniższym tekście.

Dla Czytelników mamy jeszcze niespodziankę. Ponieważ instalacja wymaga samodzielnego napisania pewnych programów, warto posłużyć się wzorcem — na przykład oprogramowaniem dla systemu MDS. Aby otrzymać kserokopię wersji źródłowej należy przesłać pod adresem redakcji:

— dużą zaadresowaną kopertę ze znaczkiem
— odcięty róg okładki z napisem mikroKLAN 7
— krótki opis posiadanego sprzętu i dotychczasowych poczynań (tylko do wiadomości redakcji).

Powyższa oferta dotyczy tylko hobbystów — osób prywatnych; ważna jest przez dwa miesiące od ukazania się numeru. Zainteresowane firmy i instytucje prosimy o kontakt bezpośredni.

Jak zainstalować CP/M

Poniższy opis dotyczy wersji CP/M 1.4. Ma on inną wielkość modułu **BDOS** i **BIOS** niż opisany w [2]. Moduł BDOS ma wielkość $D00_H$, a moduł BIOS wielkość 200_H. Dzięki temu system ten można zainstalować w systemach wyposażonych w pamięć RAM o wielkości 16 KB (a nie co najmniej 20 KB), co w przypadku konstrukcji amatorskich może być istotne. W wersji CP/M 1.4 adresy modułów dla pamięci 16 KB są następujące:

CCP — 2900_H
BDOS — 3100_H
BIOS — $3E00_H$

Adresy dla innych konfiguracji pamięci można obliczyć analogicznie jak to podano w [2]. Poniższy opis może być z powodzeniem stosowany dla innych wersji systemu CP/M. W tym przypadku, tam gdzie zostało to zaznaczone w opisie, należy korzystać z adresów podanych w [2].

Co należy mieć

1. Mikrokomputer wyposażony w:
— mikroprocesor 8080 lub odpowiednik (może być też Z80)
— terminal alfanumeryczny (np. telewizor z klawiaturą) umożliwiający wprowadzanie i wyprowadzanie znaków
— jednostkę dysków elastycznych
— pamięć RAM o pojemności min. 16 KB, umieszczoną w postaci spójnego obszaru od adresu 0.

2. Dysk elastyczny zawierający system operacyjny CP/M.

3. Środki programowe lub sprzętowe umożliwiające zapis pamięci RAM i badanie jej zawartości. Ponadto, bardzo przydatne są środki umożliwiające przygotowanie krótkich programów — do ok. 512 bajtów (np. EDYTOR i ASSEMBLER) oraz ułatwiające ich uruchamianie (np. praca krokowa i punkt wstrzymania).

4. Jeden lub dwa sformatowane, ale nie zapisane dyski elastyczne.

A oto kolejne kroki:

1. Jeżeli brak oprogramowania sterującego jednostką dysków, należy je napisać. Powinno ono umożliwić zapis i odczyt dowolnie wybranego sektora z dowolnej ścieżki dysku. Podprogram ten powinien również sygnalizować sytuacje błędne (błąd zapisu, odczytu, adresu, typu operacji itp.). Najlepiej jest umieścić go w pamięci stałej poza obszarem działania systemu CP/M.

2. Należy przygotować program odczytujący dwie pierwsze ścieżki (o adresie 0 i 1) z dysku i zapisujący je do pamięci od adresu 2880_H. Działanie tego programu należy przetestować, odczytując dwie pierwsze ścieżki z dysku o znanej zawartości — do wyzerowanej pamięci.

3. Należy przygotować program zapisujący zawartość pamięci od adresu 2880_H na dwie pierwsze ścieżki dysku i przetestować go, wykorzystując program z p. 2. Oba programy powinny być umieszczone w różnych miejscach pamięci poniżej adresu 2880_H (w przypadku nowszych wersji systemu należy zamiast adresu 2880_H w obu przypadkach użyć adresu 3380_H).

4. Jeżeli mikrokomputer nie ma programu ładowania systemu operacyjnego (np. ISIS II), to należy go przygotować i umieścić w pamięci stałej. Program ten powinien ładować do pamięci sektor nr 1 ze ścieżki nr 0 i przekazywać sterowanie do rozkazu znajdującego się w pierwszym bajcie załadowanego sektora. Jeżeli — na przykład — zamierzamy w przyszłości wykorzystywać system ISIS II lub podobny, to musimy ładować ścieżkę nr 0 od adresu 3000_H i przekazywać sterowanie pod ten adres.

5. Trzeba przygotować program ładujący system CP/M, podobny do programu z p. 2. Pewna trudność powstaje, jeżeli program z poprzedniego punktu ładuje pierwszy sektor pod adres 3000_H. Wtedy podczas ładowanie

Nie będzie polskiego komputera osobistego

Na fenomen zwany komputerem osobistym — wbrew temu, co usiłują wmówić laikom technokraci — nie składają się jedynie cudowne możliwości zaklęte w niewielkim pudełku. Pierwszorzędną rolę grają trzy zupełnie nietechniczne czynniki. Pierwszy to świadomość masowa, kształcona na powieściach science-fiction i publicystyce popołudniówek; mit elektronicznego boga zdolnego rozwiązać wszystkie problemy doczesnego życia. Drugi czynnik to łatwość dostępu: „bóstwo" można nabyć w każdym domu towarowym. Trzeci, zupełnie przyziemny aspekt to cena, zbliżona do kosztu kanapy czy naprawy samochodu.

W sferze ignorancji (pierwszy czynnik) chyba nawet przebijamy kraje zachodnie. Brak szeroko dostępnych źródeł rzetelnej, fachowej i — na dodatek — przystępnie podanej informacji sprzyja mitomanii; ba, nawet stymulowaniu snobizmu. Co gorsze — jest też jednak źródłem obaw o osobisty prestiż, o utratę prawa podejmowania „nieomylnych" decyzji. Rodzima ignorancja prowadzi do stawiania komputera w roli konkurenta, a nie narzędzia.

Rozpatrując drugi czynnik, musimy spojrzeć realnie na dotychczasową produkcję przemysłu informatycznego, który nie jest w stanie sprostać nawet zamówieniom profesjonalnych odbiorców. Masowo produkowany i sprzedawany mikrokomputer to pozbawiona najmniejszych szans mrzonka. Nie pomogą tu żadne genialne konstrukcje, programy rządowe czy nawet zakupy licencji. Przemysł informatyczny nie jest samodzielną wyspą — zależy od reszty gospodarki, o której stanie świadczy zaopatrzenie w inne „masowe" produkty.

Trzeci czynnik, czyli cena — to niestety gwóźdź do trumny. Nawet najwięksi pesymiści nie oczekują takiego wzrostu cen mebli czy usług, by sprostały one krzykliwie anonsowanym w gazetach polskim mikrokomputerom (MERITUM, ComPAN). Użycie nazwy „komputer osobisty" w stosunku do tych jednostkowo produkowanych urządzeń zakrawa na kpinę.

Warto jednak pochwalić pomysł skierowania całej produkcji MERITUM do szkół, a nie do sklepów. Zapewne musiała w tym maczać palce jakaś Federacja Ochrony Konsumenta, zabraniająca traktowania swych podopiecznych jak idiotów. Wiadomo bowiem powszechnie, że za równowartość ceny MERITUM (uwzględniając kurs czarnorynkowy) można na Zachodzie nabyć mikrokomputer o wielokrotnie lepszych parametrach.

AJP

wania program ten zostanie zniszczony. W tym przypadku musi się on najpierw przepisać pod adres niższy od 2880_H, przekazać tam sterowanie i dopiero wtedy załadować system CP/M. Program ten musi zmieścić się w 128 bajtach, ponieważ na dysku systemowym przewidziany jest dla niego tylko jeden sektor. Zawartość dysku ze ścieżki nr 0 od sektora nr 2 do końca należy załadować do obszaru pamięci od adresu 2900_H, a następnie zawartość dysku od początku ścieżki nr 1 do sektora nr 21 do obszaru pamięci od adresu 3580_H (w przypadku nowszych wersji systemu wymagających pamięci 20 KB należy załadować całe dwie pierwsze ścieżki bez pierwszego sektora pod adres 3400_H).

6. Należy przygotować moduł **BIOS**. Powinien on być skonstruowany tak jak opisano w [2], z tym że w wersji CP/M 1.4 nie są potrzebne podprogramy **LISTST** i **SECTRAN.** Jeżeli istnieją kłopoty z przygotowywaniem programów (brak programów EDYTOR i ASSEMBLER), to można także opuścić podprogramy **LIST, PUNCH** i **READER,** zwłaszcza jeżeli w systemie nie jest używana drukarka, czytnik i perforator lub inne zastępujące je urządzenia. Należy zostawić wtedy rozkaz skoku do tych podprogramów, a ich treść zastąpić rozkazem RET. Często się także zdarza, że w pamięci stałej posiadanego mikrokomputera zawarte są już podprogramy współpracy z konsolą i innymi urządzeniami peryferyjnymi. Warto je wykorzystać dopisując jedynie fragment przygotowujący parametry i odbierający wyniki, jeżeli ich specyfikacja nie jest zgodna z opisem w [2]. Do realizacji pozostałych podprogramów modułu **BIOS** można wykorzystać napisany wcześniej program współpracy z jednostką dysków (patrz p. 1).

7. Po sprawdzeniu programu z p. 5 i modułu **BIOS,** można przystąpić do utworzenia dysku z systemem CP/M. W tym celu należy:

— używając programu z p. 2 załadować do pamięci system CP/M z posiadanego dysku

— zamienić zawartość pamięci od adresu 2880_H do adresu $28FF_H$ na program przygotowany w p. 5 oraz od adresu $3E00_H$ do adresu $3FFF_H$ na nowy kod modułu BIOS przygotowany w p. 6

— zapisać na czysty dysk tak przygotowany system za pomocą programu z p. 3 (dla nowszych wersji systemu adresy 3880_H—$38FF_H$ i $4A00_H$—$4F7F_H$).

8. Przygotowany w poprzednim kroku system należy sprawdzić w sposób następujący:

— za pomocą programu z p. 4 zainicjować system. Jeżeli wszystkie poprzednie kroki były wykonane prawidłowo, to CP/M zgłasza się wyświetlając na ekranie **A>**; jeżeli system nie zgłosi się, należy jeszcze raz zbadać poprawność poprzednich kroków

— testować działanie systemu pisząc: **SAVE 1 X.COM;** po wykonaniu tej operacji system zgłasza się przez wyświetlenie na ekranie **A>**; Należy teraz napisać: **DIR;** w odpowiedzi system powinien wyświetlić na ekranie:

```
A: X .COM
A>
```

należy wtedy napisać dyrektywę kasowania: **ERA X.COM,** po wykonaniu której można ponownie dyrektywą **DIR** sprawdzić poprawność jej wykonania

— jeżeli test wypadł pomyślnie, to można skompletować dysk systemowy; wykorzystując programy z p. 2 i p. 3, należy przepisać przygotowany system na dysk zawierający jego pierwotną wersję i — wykorzystując wszystkie możliwości systemu — uzupełnić moduł **BIOS,** testując w ten sposób działanie całego systemu.

9. Nową wersję modułu **BIOS** i ewentualnie programu ładującego lub wersje systemu dla innej pamięci RAM należy przygotować tak, jak tó zostało opisane w [2]. Należy przy tym pamiętać, że dla wersji CP/M 1.4 system utworzony programem **MOVCPM** i znajdujący się również w obszarze pamięci od adresu 900_H ma program ładujący w tym samym miejscu co nowsze wersje systemu, ale moduł **BIOS** znajduje się pod adresem $1E80_H$, natomiast w docelowym systemie adresem modułu będzie adres $3E00_H+b$.

IRENEUSZ MYZIK
PIE

LITERATURA

[1] Hogan T.: Osborne CP/M User Guide. Osborne/McGraw-Hill, Berkeley, California
[2] Sobczyk M.: Dyskowy system operacyjny CP/M. INFORMATYKA, nr 2, 3, i 4, 1983.

Mimo mody (a może konieczności?) na ulepszanie i rozbudowywanie mikrokomputerów, nie radzimy nikomu przerabiać ZX81 na Mercedesa. Zamieszczony niżej program przyczynia się jednak do powiększenia komfortu pracy z tym mikrokomputerem. W „doroślejszym" ZX SPECTRUM opisane funkcje realizuje oprogramowanie producenta. ZX81 trzeba udoskonalić — by po wielogodzinnej pracy móc spokojnie zasnąć, ze świadomością, że nasz program utrwalony na kasecie nie okaże się nazajutrz tylko iluzją.

Możliwość przejrzenia kasety jest cenna nie tylko dla bałaganiarzy. Pozwala również sprawdzić, czy nasze „biblioteczne" programy są nadal w porządku. Szkoda tylko, że nikt — jak dotąd — nie wymyślił praktyki mniej czasochłonnej.

Program VTL dla ZX81

Nazwa programu utworzona została od słów angielskich VERIFY (sprawdzanie) i TAPE LIST (wydruk zawartości taśmy). VTL składa się ze 144 bajtów kodu maszynowego mikroprocesora Z80 i realizuje dwie funkcje:

VERIFY: RAND USR 16514 — weryfikacja poprawności nagrania magnetofonowego przez porównanie kodów odczytywanych z taśmy, z zawartością pamięci mikrokomputera (z pominięciem zmiennych systemowych). Komunikaty: 0/0 — nagranie poprawne; E/0 — błąd (ang. error).

TLIST: RAND USR 16620 — wyświetla na ekranie zawartość taśmy (tytuły kolejnych programów). Każdorazowo po odczytaniu z taśmy tytułu programu przez kilka sekund wyświetlany jest on na ekranie TV, po czym program VTL kontynuuje czytanie taśmy. Pracę programu można w dowolnym momencie przerwać klawiszem **BREAK.** Pojawiające się na ekranie znaki graficzne, a zwłaszcza czarna kreska, świadczą o uszkodzeniu zapisu na taśmie lub niewłaściwym ustawieniu potencjometru głośności magnetofonu.

```
16 FF CD E7 02 CD A3 3A
1B FB C9 0E 01 06 00 3E
7F DB FE D3 FF 1F 30 42
17 17 38 21 10 F1 F1 BA
30 46 CD A3 3A CB 42 79
28 44 17 30 F5 21 09 40
42 CD A3 3A 79 BE 20 24
CD FC 01 18 F3 D5 1E 94
06 1A 1D DB FE 17 CB 7B
7B 38 F5 10 F5 D1 20 04
FE 56 30 B9 3F CB 11 30
B4 C9 CF 0C D5 EB 21 7D
40 37 ED 52 EB D1 30 D0
CF 0D 16 FE 18 94 CB 77
28 02 3E 80 D7 CB 11 30
A9 3E 76 D7 D5 CD 07 02
1E 96 CD 46 0F D2 A6 03
10 F8 1D 20 F5 D1 18 DC
```

Wydruk 1

Program VTL przedstawiono na wydruku 1 w postaci listy bajtów (w kodzie szesnastkowym). Do jego wprowadzenia należy wykorzystać program przedstawiony na wydruku 2. Wpisujemy go z klawiatury i uruchamiamy przyciskiem **RUN.** Możemy teraz wprowadzać kolejne bajty programu VTL z wydruku 1. Jednorazowo można wprowadzać dowolną liczbę bajtów (najwygodniej wierszami), przy czym należy zwrócić uwagę na to, że:

— liczba wprowadzonych przed każdym wciśnięciem klawisza NEWLINE cyfr szesnastkowych musi być parzysta — pomiędzy cyframi nie może występować spacja.

```
 1 LET ABCDEFG=0+0+0+0+0+0+0+0+0+0+0+0+0+0+0+0
 2 POKE 16510,0
 3 POKE 16513,234
 4 LET B=16514
 5 INPUT A$
 6 FOR J=0 TO LEN A$-1 STEP 2
 7 POKE B+J/2,CODE A$(1)*16+CODE A$(2)-476
 8 LET A$=A$(3 TO )
 9 NEXT J
10 LET B=B+J/2
11 IF B<16658 THEN GOTO 5
```

Wydruk 2

Program VTL pamiętany jest w wierszu: 0 REM... . Ponieważ w programie VTL występuje kod NEW LINE (76 HEX=118 DEC), zatem w przypadku automatycznego wydruku wyświetlany jest tylko fragment wiersza 0 REM... Niedogodność tę można usunąć instrukcją:

POKE 16419, 1 [NEW LINE]

Program VTL należy przetestować z magnetofonem. Jeśli nie popełniliśmy błędu przy jego wprowadzaniu, to wiersze 2—11 (wydruk 2) można usunąć, a na ich miejsce wprowadzić program z wydruku 3.

```
 3 PRINT """VTL""  144 BAJTY";TAB 0;"ŁADOWAC OD ADRESU?"
10 INPUT A
11 IF A<8192 THEN LET A=16514
12 FAST
15 LET L=A+11
20 LET H=INT (L/256)
25 LET L=L-256*H
30 FOR I=0 TO 143
35 POKE A+I,PEEK (16514+I)
40 NEXT I
45 POKE A+6,L
50 POKE A+7,H
55 POKE A+35,L
60 POKE A+36,H
65 POKE A+50,L
70 POKE A+51,H
73 PRINT AT 5,0;"USR ";A;" = VERIFY";TAB 0;"USR ";A+106;" = TLIST"
74 SLOW
75 STOP
80 SAVE "VTL"
85 RUN
```

Wydruk 3

Program 3 służy do przemieszczania programu VTL w pamięci. Najwygodniejszymi obszarami do przechowywania tego typu programów są:

— dla komputerów z pamięcią 64 KB — obszar między 8 a 16 K (adresy 8191—16383 DEC)

— dla komputerów z pamięcią od 1 do 16 KB — obszar powyżej zmiennej systemowej RAMTOP.

Przykładowo, dla pamięci 16 KB:

```
POKE 16389, 127     [NEW LINE]
POKE 16388, 112     [NEW LINE]
NEW                 [NEW LINE]
LOAD "VTL"          [NEW LINE]
```

Adres, pod którym zostanie załadowany program VTL:

127 * 256 + 112 = 32624

Na pytanie: **LADOWAC OD ADRESU?** należy wprowadzić:

32624 [NEW LINE]

Po załadowaniu programu wyświetlony zostanie komunikat:

USR 32624 =VERIFY
USR 32730 =TLIST

Całość (VTL + program 2) należy zapisać na taśmie instrukcją:

RUN 80 [NEW LINE]

Przed autoweryfikacją nagrania programu VTL, należy wykonać instrukcję **CLEAR**, gdyż po zapisie program zostaje uruchomiony (wiersz 85 RUN), a pamięć mikrokomputera zawiera zmienne i ich wartości, których nie ma na taśmie (zostały skasowane w momencie wydania rozkazu RUN 80.

JAN RABIZO
Agencja Komputerowa „AMICO"
Sosnowiec, tel. 699—649

Mikroprocesor Z80 jest „sercem" wielu popularnych mikrokomputerów. Tym ciekawsze wydaje się doniesienie o dodatkowych rozkazach, rozszerzających powszechnie znany zestaw. Dotyczą one jednak w znacznej mierze operacji wykonywanych na rejestrach IX i IY. Rejestry te mogłyby być bardzo użyteczne, gdyby nie fakt, że ich stosowanie wymaga często 4-bajtowych kodów operacji. Pochłania to pamięć programu i czas procesora. Większość opisanych niżej kodów wymaga tylko dwóch bajtów. Być może przyczyni się to do powszechniejszego stosowania rejestrów IX i IY.

Niepublikowane rozkazy mikroprocesora Z80

Katalogi firm produkujących mikroprocesory Z80 podają 158 grup rozkazowych tego układu. Przeglądając listę rozkazów uporządkowaną według kodów szesnastkowych, można trafić na puste miejsca. Nasuwa się więc pytanie, co zrobi mikroprocesor po napotkaniu jednej z niezdefiniowanych sekwencji bajtów. Okazuje się, że w większości przypadków działanie układu jest sensowne.

Podane dalej informacje sprawdzono na mikroprocesorach firm: NEC, MOSTEK, ZILOG i RFT. Przyjęto następujące dodatkowe oznaczenia:

HX — bardziej znaczący bajt rejestru IX
LX — mniej znaczący bajt rejestru IX
HY — bardziej znaczący bajt rejestru IY
LY — mniej znaczący bajt rejestru IY
F3 — trzeci bit rejestru wskaźników,
F5 — piąty bit rejestru wskaźników.

Rozkaz **SLI** (Shift Left & Increment) ma działanie podobne do SLA, z tym, że na najmniej znaczący bit rejestru podawana jest jedynka. Stan rejestru wskaźników F odpowiada wynikowi operacji, podobnie jak dla rozkazu SLA. Rozkaz **SLI n** ma kod CB3x, gdzie x przyjmuje wartości zależnie od operandu (tabela 1).

Tebela 1

n	A	B	C	D	E	H	L	(HL)
x	7	0	1	2	3	4	5	6

Grupa rozkazów występujących po kodach DD lub FD zawiera najwięcej rozszerzeń w stosunku do listy publikowanej w katalogach. Wszystkie rozkazy (z wyjątkiem EX DE,HL) dotyczące rejestrów H, L oraz pary HL mają swe odpowiedniki dla rejestrów HX, LX, HY, LY. Bity w rejestrze wskaźników przy wykonywaniu tych rozkazów ustawiane są identycznie jak dla standardowych instrukcji. W tabeli 2 podano tylko wersję rozkazów dla rejestru IX. Rozkazy dotyczące IY uzyskamy zastępując DD, w kodzie rozkazu, przez FD.

Rozkazy znalezione w grupie „po DDCB/FDCB" wydają się być ubocznymi efektami działania standardowych instrukcji mikroprocesora, jednakże ich wykorzystanie może być czasem użyteczne. Argumentem rozkazów jest zawsze bajt o adresie IX+d (lub IY+d), natomiast wynik jest przesyłany zarówno do bajtu o adresie IX+d (lub IY+d), jak i do jednego z rejestrów wewnętrznych.

Na przykład — rozkaz RLC(IX+ +01),B (kod DDCB0100) powoduje wykonanie standardowego rozkazu RLC (IX + 01) (kod DDCB010E) oraz przesyłanie wyniku operacji do rejestru B. Analogicznie działają pozostałe rozkazy, tzn. RRC, RLC, RR, RL, SLA, SRA, SLI (również!), SRL oraz SETn i RESn.

Do jednego z rejestrów wewnętrznych jest przesyłana zawsze zawartość wpisywana do bajtu (IX+d) lub (IY+d) po wykonaniu operacji (również w wyniku operacji SET i RES).
Format rozkazów tej grupy jest następujący:

DD CB dd XX lub FD CB dd XX

gdzie XX jest jednym z kodów podanych w tabeli 3.

Rozkazy o bajcie XX równym 01nnnrrr (odpowiadające rozkazom BIT grupy „po CB") działają naturalnie tylko na bitach pamięci zewnętrznej (IX+d) lub (IY+d) — identycznie jak standardowe rozkazy BIT n, (IX+d) lub BIT n, (IY+d).

Napotkanie po bajcie DD lub FD kodu niewymienionego powyżej powoduje zignorowanie bajtu DD lub FD i wykonanie zwykłego rozkazu z listy firmowej.

Na przykład — DDEB powoduje „tylko" EX DE,HL. DD 01 23 45 działa identycznie jak 01 23 45 (LD BC, 4523), a DD ED 7A jak ED 7A (ADC HL,SP).

Bity F5 i F3 w rejestrze wskaźników odzwierciedlają stan odpowiednio trzeciego i piątego bitu wyniku operacji arytmetycznej lub logicznej. Działanie tych bitów sprawdzono na rozkazach operujących na danych ośmiobitowych:

ADD, ADC, SUB, SBC, OR, XOR, AND, CP,

oraz

RLC, RRC, RL, RR, SLA, SRA, SLI, SRL

a także

IN r, (C) — włącznie z IN F, (C).

Przykładowo — niech A = 01110001. Po wykonaniu rozkazu RLC A (kod CB 07) stan akumulatora będzie następujący:

A = 11100010

Bity w rejestrze wskaźników: F5=1, F3 = 0.

Najstarszy, siódmy bit rejestru R nie ulega zmianie przy autoinkrementacji tego rejestru (inkrementowane są tylko bity 0...6), można go więc wykorzystać do pamiętania dowolnych informacji (w wyniku działania rozkazu LD R,A).

Tabela 2

Kod	Proponowana nazwa mnemoniczna	Działanie
DD 24	INC HX	HX←HX+1
DD 25	DEC HX	HX←HX−1
DD 26nn	LD HX, NN	HX←nn
DD 2C	INC LX	LX←LX+1
DD 2D	DEC LX	LX←LX−1
DD 2Enn	LD LX, NN	LX←nn
DD 44	LD B, HX	$r_1 \leftarrow r_2$
DD 45	LD B, LX	
DD 4C	LD C, HX	
DD 4D	LD C, LX	
DD 54	LD D, HX	
DD 55	LD D, LX	
DD 5C	LD E, HX	
DD 5D	LD E, LX	
DD 6x	LD HX, r	$HX \leftarrow r_2$

r	B	C	D	E	HX	LX	A
x	0	1	2	3	4	5	7

DD 6x	LD LX, r	$LX \leftarrow r_2$

r	B	C	D	E	HX	LX	A
x	8	9	A	B	C	D	F

DD 7C	LD A, HX	A←HX
DD 7D	LD, A, LX	A←LX
DD 84	ADD A, HX	A←A+HX
DD 85	ADD A, LX	A←A+LX
DD 8C	ADC A, HX	A←A+HX+CF
DD 8D	ADC A, LX	A←A+LX+CF
DD 94	SUB HX	A←A−HX
DD 95	SUB LX	A←A−LX
DD 9C	SBC A, HX	A←A−HX−CF
DD 9D	SBC A, LX	A←A−LX−CF
DD A4	AND HX	A←A∧HX
DD A5	AND LX	A←A∧LX
DD AC	XOR HX	A←A÷HX
DD AD	XOR LX	A←A÷LX
DD B4	OR HX	A←A∨HX
DD B5	OR LX	A←A∨LX
DD BC	CP HX	A−HX
DD BD	CP LX	A−LX

Uwaga: ÷ oznacza różnicę symetryczną (ang. Exclusive OR)

Tabela 3

Mnemonik	Bajt XX		
RLC (IX+d), r	00000rrr	rrr =	000 B
RRC (IX+d), r	00001rrr		001 C
RL (IX+d), r	00010rrr		010 D
RR (IX+d), r	00011rrr		011 E
SLA (IX+d), r	00100rrr		100 H
SRA (IX+d), r	00101rrr		101 L
SLI (IX+d), r	00110rrr		111 A
SRL (IX+d), r	00111rrr		110 (IX+d)
RES n, (IX+d), r	10nnnrrr		
SET n, (IX+d), r	11nnnrrr		

PAWEŁ MAĆKÓW
Lubin

LITERATURA

[1] Weitere U-800-Befehle — Radio, Fernsehen, Elektronik 11, 1983, str. 726,
[2] Unbekannte Z80-Befehle — Elektronik (München), 14, 1980, str. 83
[3] Z80 Assembly Language Programming. L. A. Leventhal — Osborne/McGraw-Hill, 1979.

Przy uruchamianiu programu najwięcej czasu pochłania wyszukiwanie błędów. Najbardziej irytujące jest jednak — poprawianie ich. Nawet gdy program w wersji źródłowej przechowywany jest w pamięci operacyjnej (nie traci się czasu na ładowanie z dysku lub — o zgrozo! — z taśmy papierowej), czas, w którym program jest powtórnie asemblowany dłuży się w nieskończoność. Problem ten wydaje się eliminować system LIC. Dodatkową zaletą systemu jest asemblacja w trybie konwersacyjnym. Pozwala to na bieżąco korygować pomyłki wynikłe przy zapisywaniu programu. Niestety, nie jesteśmy w stanie zamieścić programu; wszystkich zainteresowanych odsyłamy więc do autora systemu.

LIC
wspomaganie projektowania programów

System LIC (ang. label, instruction, comment) jest przeznaczony do konwersacyjnego projektowania programów w języku ASSEMBLER 8080. System LIC umożliwia asemblację, edycję oraz uruchamianie programów. Proces asemblacji jest jednoprzebiegowy, odwracalny; generuje relokowalny kod wynikowy. Językowo zorientowany edytor (ang. symbolic debugger) jest dostępny po asemblacji, w fazie uruchamiania programu. W konwencjonalnych systemach projektowania modyfikacja programu źródłowego wiąże się z koniecznością jego ponownej asemblacji. System LIC umożliwia bezpośrednią modyfikację kodu wynikowego. Edycja programów wynikowych jest szybka (wykonywana w pamięci operacyjnej) i wygodna (DISASEMBLER zapewnia, że kod wynikowy jest widziany przez użytkownika w postaci źródłowej). Językowo zorientowany EDYTOR, ASEMBLER i DISASEMBLER zapewniają szybką integrację sprzętu z oprogramowaniem.

System LIC jest więc szczególnie atrakcyjny dla użytkowników mikrokomputerów nie wyposażonych w pamięci dyskowe. System LIC spełnia wiele funkcji systemu operacyjnego i może być wykorzystany jako program MONITOR.

ASEMBLER w procesie analizy programu źródłowego generuje kod wynikowy, listę etykiet oraz listę komentarzy. Etykiety i komentarze są przechowywane w pamięci operacyjnej i mogą być modyfikowane konwersacyjnie. DISASEMBLER w procesie syntezy kodu wynikowego, symbolicznych etykiet oraz komentarzy generuje program źródłowy. Program źródłowy można asemblować z dowolnego urządzenia wejściowego lub disasemblować na dowolne urządzenie wyjściowe. Podobnie można wprowadzać lub wyprowadzać kod wynikowy, listę etykiet oraz listę komentarzy. Fizyczne urządzenia wejścia-wyjścia są przyporządkowane strumieniom logicznym. Strumieniowi o numerze 0 przypisano klawiaturę i monitor ekranowy. Liczba strumieni wejścia-wyjścia zależy od konfiguracji systemu mikrokomputerowego.

Konwersacyjna asemblacja pozwala na szybką eliminację błędów składniowych. System LIC sygnalizuje miejsce oraz numer błędu. Asembler wprowadza identyfikację (wartością wskaźnika wprowadzania) wierszy źródłowych oraz produktów asemblacji. Identyfikacja obiektów jest wykorzystywana w procesie edycji. Wskaźnik wprowadzania wyznacza miejsce ładowania kodu i jest zawsze określony przed analizą wiersza źródłowego. Wskaźnik wprowadzania stanowi jednocześnie numer wiersza źródłowego, wartość etykiety, adres rozkazu, danej oraz numer komentarza. Adresowi pamięci operacyjnej przypisuje co najwyżej jeden rozkaz (daną), jedną etykietę, jeden komentarz, a w konsekwencji — jeden wiersz źródłowy.

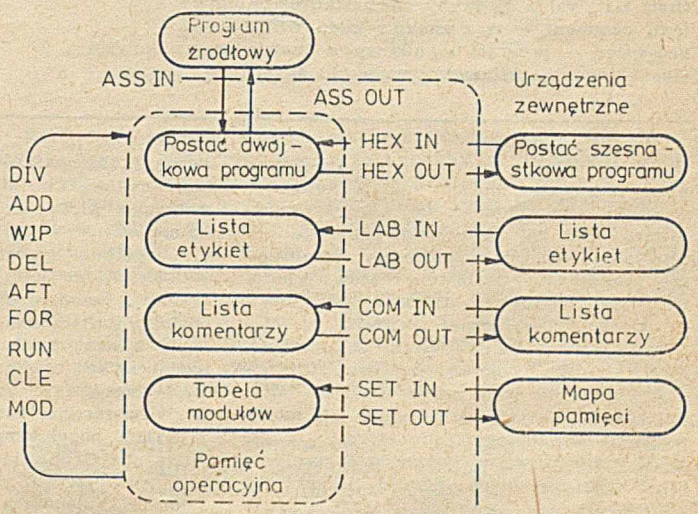
Programy projektowane w systemie LIC składają się ze spójnych obszarów danych i instrukcji. Obszary te nazwano modułami. Modułowa budowa programów umożliwia fragmentaryczną asemblację oraz organizację prostej biblioteki. Moduły stanowiące program można asemblować w dowolnej kolejności. Uruchomione moduły można składować w postaci wynikowej. System LIC dopuszcza składanie programów z modułów źródłowych oraz wynikowych. EDYTOR umożliwia dzielenie, łączenie, usuwanie i relokację modułów wynikowych. Relokacja wiąże się ze zmianą wartości etykiet, zmianą lokalizacji rozkazów (danych) oraz zmianą numeracji komentarzy. W związku ze zmianą wartości etykiet, ulegają zmianie argumenty instrukcji 3-bajtowych, odwołujących się do zmienianych etykiet. Zmiana argumentów rozkazów 3-bajtowych następuje przy przejściu z trybu relokowalnego do trybu absolutnego (MOD). Tryb relokowalny skraca czas edycji oraz umożliwia wykorzystanie niezdefiniowanych jeszcze (ang. forwards) etykiet. Przejście z trybu relokowalnego do absolutnego można traktować jako dopełnienie asemblacji lub konsolidację modułów.

Opis modułu wynikowego składa się z granicy górnej, granicy dolnej obszaru pamięci oraz jednoliterowej nazwy. Nazwy modułów umożliwiają wygodną identyfikację w procesie edycji. Lista opisów modułów (SET OUT) tworzy mapę pamięci operacyjnej. Moduły można wprowadzać (ASS IN) lub definiować (HEX IN, LAB IN, COM IN, SET IN). W pierwszym przypadku produkty asemblacji tworzą treść modułu. W drugim przypadku zawartość modułu jest określana stopniowo poprzez wprowadzanie zawartości pamięci, etykiet, komentarzy i parametrów modułu. Nieznany program binarny można — przykładowo — wprowadzić do pamięci (HEX IN), opisać etykietami (LAB IN), komentarzami (COM IN), a następnie wyprowadzić w postaci źródłowej (ASS OUT). Standardowe etykiety symboliczne są generowane w następstwie dyrektywy SET IN.

Moduły są składowane w postaci źródłowej (ASS OUT) lub w postaci produktów asemblacji (HEX OUT, LAB OUT, COM OUT, SET OUT). Obydwie metody są równoważne.

Program wynikowy jest składowany w kodzie absolutnym (HEX OUT) lub w postaci relokowalnej (HEX OUT, SET OUT). Program wynikowy jest ładowany w miejsce robocze, które zajmował przed składowaniem. Moduły programu relokowalnego można przesunąć na miejsce docelowe (AFT, FOR). Modułom relokowalnym można przypisać symboliczne odwołania wewnętrzne (LAB). Moduły absolutne dopuszczają jedynie symboliczne odwołania wewnętrzne.

System LIC posiada dwa wejścia. Pierwsze służy do inicjowania systemu, natomiast drugie można stosować jako wyjście z programu użytkownika. Drugie wejście oraz procedury monitorowania, dowolnie definiowane przez użytkownika, zastępują w systemie LIC konwencjonalne punkty wstrzymania (ang. break point). Udostępnia ono wszystkie dyrektywy systemu. Wprowadzenie poprawek nie wymaga ponownej asemblacji programu.



Przepływ informacji w systemie LIC

Programy źródłowe oraz wynikowe projektowane w systemie LIC zachowują standard firmy INTEL. Program systemu LIC przeznaczony dla mikrokomputerów wykorzystujących mikroprocesor 8080 zajmuje 6 KB pamięci ROM. Implementacja programu jest uwarunkowana jedynie konfiguracją urządzeń zewnętrznych.

```
LIC-system   := (command)+
command      := / ASS IN  nb first,§
                  (data-space/instr-space)
                / HEX IN  nb
                  (hex-space)
                / LAB IN  nb
                  (label:value)+
                / COM IN  nb
                  (value;comment)+
                / SET IN  nb
                  (§,first,last)+
                / ASS OUT nb §,first,last
                / HEX OUT nb
                  (§,first,last)+
                / LAB OUT nb
                / COM OUT nb
                / SET OUT nb
                / DIV §,value,%
                / ADD §,%
                / WIP §
                / DEL §,first,last
                / AFT value,§
                / FOR value,§
                / RUN value
                / CLE
                / MOD
data-space   := (     declaration
                /     declaration;comment
                / label:declaration;comment )+
instr-space  := (     instruction
                /     instruction;comment
                / label:instruction;comment )+
declaration  := / DB ('text'/ const )
                / DW const
                / DS const
instruction  :=  CMA/CMC/DAA/DI/EI/HLT/NOP/PCHL/
                 RAL/RAR/RC/RET/RM/RNC/RNZ/RP/
                 RPE/RPO/RRC/RZ/SPHL/STC/XCHG/XTHL/
                (ADC/ADD/ANA/CMP/DCR/
                 INR/ORA/SBB/SUB/XRA) (A/B/C/D/E/H/L/M) /
                (INX/DCX/DAD) (B/D/H/SP) /
                (POP/PUSH) (B/D/H/PSW) /
                (LDAX/STAX) (B/D) /
                 MOV (A/B/C/D/E/H/L/M),(A/B/C/D/E/H/L/M) /
                 MVI (A/B/C/D/E/H/L/M),    const /
                (ACI/ADI/ANI/CPI/IN/ORI/
                 OUT/RST/SBI/SUI/XRI)      const /
                (CALL/CC/CM/CNC/CNZ/CP/CPE/CPO/CZ/
                 JCC/JM/JMP/JNC/JNZ/JP/JPE/JPO/JZ/
                 LDA/LHLD/SHLD/STA) label
                 LXI (B/D/H/SP),    label
```

Składnię systemu LIC podano poniżej. Oto objaśnienia zastosowanych zapisów. Metasymbole : = , (,) , + , / oznaczają kolejno: definicję, nawias otwierający, nawias zamykający, rozkaz powtórzenia skończoną liczbę razy, lub.

first, last, value — liczby szesnastkowe 16-bitowe
text, comment — ciąg znaków kodu ASCII
hex-space — program wynikowy w kodzie szesnastkowym
label — symbol alfanumeryczny (do 5 znaków)
const — liczba kodowana dwójkowo, ósemkowo, dziesiętnie lub szesnastkowo
§, % — dowolny znak kodu ASCII
nb — numer strumienia (urządzenia zewnętrznego)

ASS IN — asemblacja tekstu źródłowego. Określenie wskaźnika wprowadzania wartością **first**. Utworzenie modułu wynikowego o nazwie § oraz granicy górnej **first**. Moduł jest wypełniany w miarę wprowadzania kolejnych wierszy źródłowych

ASS OUT — disasemblacja fragmentu **first-last** modułu wynikowego o nazwie §

HEX IN — ładowanie pamięci operacyjnej kodem szesnastkowym

HEX OUT — składowanie zawartości fragmentu **first-last** pamięci operacyjnej w kodzie szesnastkowym

SET IN — definicja modułu wynikowego o nazwie § oraz granicach **first, last**. Obszar **first-last** traktowany jest jako spójny obszar (danych, instrukcji) w zależności od przyjętej nazwy modułu (znak nieliterowy, litera). W przypadku spójnego obszaru instrukcji sporządzana jest lista argumentów instrukcji 3-bajtowych. Generator etykiet przypisuje argumentom standardowe nazwy

SET OUT — składowanie parametrów §, **first, last** wszystkich modułów wynikowych

LAB IN — definicja etykiet o nazwach **label** i wartościach **value**. Jeżeli etykieta o wartości **value** istniała, to przyporządkowana jej zostanie nowa nazwa **label**. Jeżeli etykieta o nazwie **label** istniała, to przyporządkowana jej zostanie nowa wartość **value**

LAB OUT — listowanie etykiet w kolejności alfanumerycznej

COM IN — definicja komentarzy o numerach **value** i treściach **comment**. Poprzednia treść komentarza, o ile istniała, jest gubiona

COM OUT — listowanie komentarzy w kolejności numeracji

DIV — podział modułu § wartością **value** na moduły § i %

ADD — połączenie modułów stycznych §, % w moduł §

WIP — usunięcie modułu §

DEL — usunięcie fragmentu **first-last** modułu §

AFT — przesunięcie modułu § tak, aby jego granica górna pokryła się z wartością **value**. Jeżeli wartość **value** dotyczy obszaru zajętego przez inny moduł %, to moduł § zostanie wstawiony do wnętrza %. Dolna granica modułu % przesunie się, a moduł § przestanie istnieć

FOR — przesunięcie modułu § tak, aby jego dolna granica pokryła się z wartością **value**. Jeżeli wartość **value** dotyczy obszaru zajętego przez inny moduł %, to moduł § zostanie wstawiony do wnętrza %. Górna granica modułu % przesunie się, a moduł § przestanie istnieć

RUN — rozpoczęcie wykonywania programu od adresu **value**

CLE — ustawienie stanu początkowego systemu LIC

MOD — zmiana trybu relokowalny-absolutny. Tryb relokowalny oznaczany jest gwiazdką, a tryb absolutny krzyżykiem. Znaki te wyprowadzane są na monitor przy każdorazowym zgłoszeniu się systemu. Dyrektywy **ASS IN, WIP, DEL, FOR, AFT** dozwolone są tylko w trybie relokowalnym. Dyrektywy **RUN, MEX OUT** dozwolone są tylko w trybie absolutnym.

BOGDAN MICHALAK
Warszawa
tel. sł.: 23-70-81 w. 237
tel. dom.: 39-52-96

● W związku z często zadawanym nam pytaniem, gdzie można nabyć komputer osobisty, odpowiadamy: w każdym domu towarowym... na zachód od Łaby. Zainteresowanym zakupem polecamy opracowanie „Komputery Osobiste" rozpowszechnione przez Branżowy Ośrodek Informacji Naukowej i Technicznej (BOINTE, ul. Marynarska 10, 02-674 Warszawa, tel. 47-07-62). Można w nim znaleźć opisy większości liczących się na rynkach zachodnich komputerów osobistych, porównanie ich możliwości, a nawet tegoroczne ceny.

● Przedsiębiorstwo Polonijno-Zagraniczne „KAREN" zapowiada montaż i sprzedaż mikrokomputera zgodnego z IBM PC. W komplecie przewidziano dwa napędy dysków elastycznych 5.25" oraz drukarkę graficzną (kolorową!). Kontakt: Warszawa, ul. Krochmalna 2 m. 127, tel. 20-11-42.

● Uwaga wielbiciele gier komputerowych! Zamieszczona w INFORMATYCE propozycja wysyłania programów gier w języku FORTRAN nie jest już, niestety, aktualna. W tej sytuacji przekazujemy radę Leszka Drożdża z Rzeszowa polecającego w swym liście książkę Lecha Pijanowskiego „Przewodnik gier", zawierającą opisy wielu gier strategicznych.

● W czwartym mikroKLANIE w rubryce FORTHA popełniono błąd — podając, że bitwa pod Grunwaldem odbyła się we wtorek. Autor tekstu zapomniał o późniejszej reformie kalendarza. Sądzimy jednak, że zwolennicy FORTHA bez trudu zmodyfikują przedstawiony program.

W poniższym tekście nie odkrywamy Ameryki. Dla doświadczonych konstruktorów wymyślenie podanych niżej rozwiązań nie przedstawia najmniejszego problemu. Jednak — jak wskazują listy i telefony do redakcji — wśród czytelników mikroKLANU wielu stawia dopiero pierwsze kroki...

Biorąc pod uwagę ustawiczne niedostatki rynkowe, warto przekonać nowicjuszy, że brak jakiegoś układu nie jest jeszcze powodem do załamywania rąk. Wystarczy odrobina inwencji i zawsze znajdzie się jakieś wyjście. Czasem trochę gorsze, innym razem... znacznie lepsze.

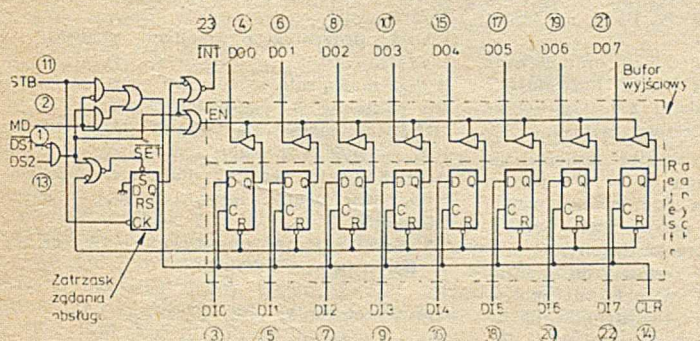
8212 zastępuje 8255 w trybie pierwszym

Układ 8212 wykorzystywany jest najczęściej jako bufor wejściowy lub wyjściowy. Zawarte w jego strukturze zatrzaski pozwalają na zapamiętanie informacji wysłanej przez procesor (wyjście) lub urządzenie zewnętrzne (wejście). Ważną cechą układu jest trzystanowe wyjście (tzw. trzeci stan, nazywany też stanem dużej impedancji, polega na tym, że układ nie wymusza na wyjściu żadnego stanu logicznego — tak jakby nie był w ogóle podłączony), co pozwala na współpracę z magistralą danych systemu mikroprocesorowego.

Konstruktorzy często zbyt pochopnie rezygnują z wykorzystania układu 8212, stosując bardziej „wyrafinowany" układ 8255. Tymczasem w przypadku, gdy w systemie stosowane jest pojedyncze strobowane wejście lub wyjście — z powodzeniem można wykorzystać układ 8212, który jest sporo tańszy od 8255. Przyjęcie takiego rozwiązania ma następujące zalety:

- 8212 wykonany jest w technologii TTL, a więc ma dziesięciokrotnie większą obciążalność wyjść niż 8255
- 8212 nie wymaga programowania i pracuje w żądanej konfiguracji od momentu włączenia zasilania.

Do wad układu należy brak możliwości programowego stwierdzenia źródła przerwania (może to spowodować konieczność zastosowania kontrolera przerwań).

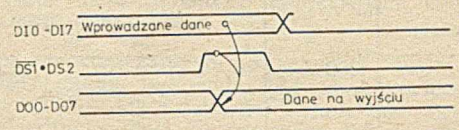


Rys. 1. Struktura logiczna układu 8212

Na rysunku 1 przedstawiono wewnętrzną organizację logiczną układu 8212. Przepływ danych sterowany jest za pomocą wejść: MD (wybór trybu pracy), STB (sygnał strobujący) i DS1/, DS2 (linie wybierające). Układ jest wybierany (bufory wyjściowe nie są w trzecim stanie), gdy na wejście DS1/ podany zostanie poziom niski i na wejście DS2 — poziom wysoki lub gdy na wejście MD podawany jest poziom wysoki. W tym drugim przypadku układ pracuje w sposób ciągły jako wyjście, a dane wpisywane są do zatrzasków sygnałami z linii wybierających. Stan wejścia STB nie ma wtedy wpływu na przepływ danych.

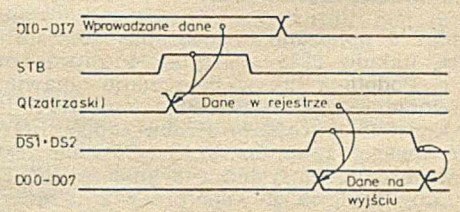
Jeśli na wejście MD podany zostanie poziom niski, dane do zatrzasków wpisywane są sygnałem STB, niezależnie od stanu linii wybierających układ. Linie wyboru sterują jedynie buforami wyjściowymi. Układ pracuje więc w trybie wejściowym.

Zatrzaski mogą zostać wyzerowane w dowolnym momencie przez podanie niskiego poziomu na wejście CLR/.



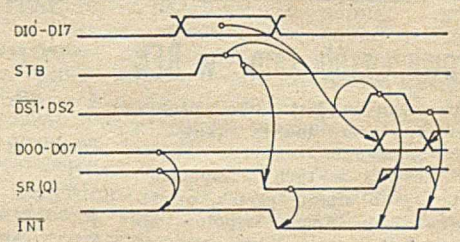
Rys. 2. Przebiegi czasowe dla trybu określonego przez MD=1

Na rysunku 2 podane zostały przebiegi czasowe przy wyborze trybu pracy przez podanie na wejście MD poziomu wysokiego. Na rysunku 3 pokazano natomiast przebiegi czasowe, gdy na MD podany jest poziom niski.



Rys. 3. Przebiegi czasowe dla trybu określonego przez MD=0

Jeżeli na wejściu STB nastąpi zmiana poziomów z wysokiego na niski, to wyjście przerzutnika SR zostanie ustawione w stan 0. Powoduje to wymuszenie niskiego poziomu na wyjściu INT/. Przerzutnik SR ustawiany jest ponownie w stan 1, gdy na wejście CLR/ podany zostanie poziom niski lub gdy na DS1/ podane zostanie 0 i na DS2 — 1. Na rysunku 4 pokazano przebiegi czasowe związane z generacją sygnału INT/.

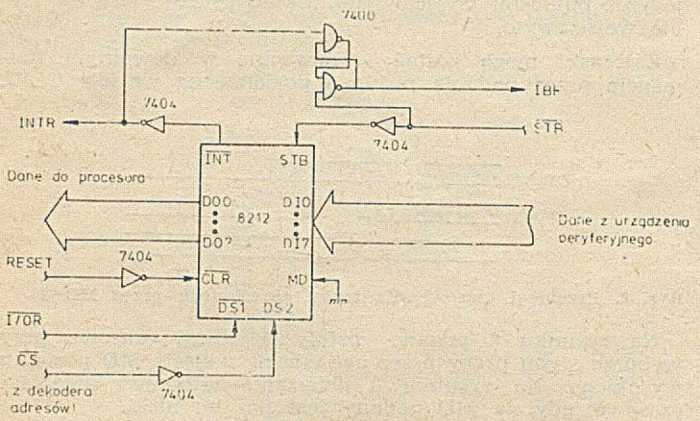


Rys. 4. Zależności czasowe dla sygnału generacji przerwań

Za pomocą kilku dodatkowych bramek można utworzyć konfigurację odpowiadającą współpracy z układem 8255 w trybie 1, gdy został on zaprogramowany jako wejście. Przykładowe rozwiązanie pokazano na rysunku 5. Przebiegi czasowe różnią się tylko dla sygnału zgłoszenia przerwania INTR (aktywny w stanie 1). Dla układu 8255 przechodzi on w stan pasywny 0 — po przejściu strobu odczytu I/OR/ w stan 0 (aktywny). W proponowanej konfiguracji sygnał INTR przechodzi w stan 0 dopiero z chwilą zakończenia strobu odczytu. W praktyce nie ma to większego znaczenia, gdyż zakończenie procedury obsługi przerwania wymaga z reguły wykonania jeszcze kilku instrukcji po odczycie danej.

Brak możliwości bezpośredniego odczytywania statusu praktycznie eliminuje programowy sposób realizacji współpracy (bez wykorzystywania przerwań). Jest to chyba najpoważniejsza wada przedstawionego rozwiązania, gdyż dodanie dodatkowego układu z trzystanowym wyjściem (np. 8216) znacznie podraża koszty konstrukcji. Warto jed-

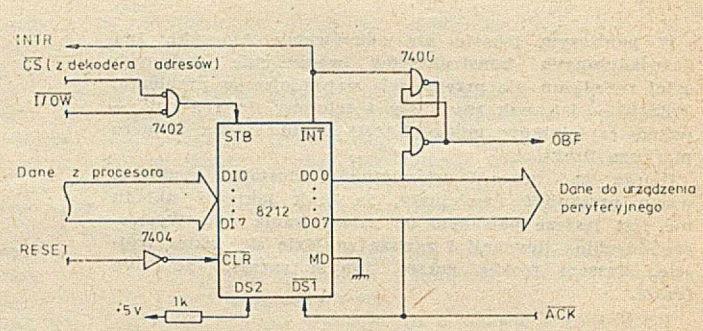
nak dodać, że stosowanie pierwszego trybu pracy dla układu 8255 jest uzasadnione, gdy zakłada się korzystanie z przerwań.



Rys. 5. Układ strobowanego wejścia

Na rysunku 6 pokazano rozwiązanie pozwalające przybliżyć pracę układu 8255 w trybie 1 zaprogramowanego jako wyjście. Podobnie jak w poprzednio omawianym rozwiązaniu, zgłoszenie przerwania sygnałem INTR jest likwidowane dopiero po wycofaniu strobu zapisu I/OW/. Warto też zwrócić uwagę, że przerwanie zgłaszane jest natychmiast po wyzerowaniu systemu sygnałem RESET. Jeżeli nie zamierzamy realizować współpracy z urządzeniem peryferyjnym od samego początku pracy systemu, należy przed wykonaniem przez procesor rozkazu EI ustawić w kontrolerze przerwań odpowiednią maskę. Innym sposobem może być wykorzystanie, faktu, że zgłoszenie przerwania zostanie wycofane po wykonaniu dowolnego zapisu do układu 8212. Metoda ta może być jednak wykorzystana tylko w tych przypadkach, gdy istnieje pewność, że urządzenie peryferyjne nie wprowadzi „fałszywej" danej.

Układ 8255 pracujący w trybie 1 steruje linie wyjściowe niezależnie od stanu podawanego na wejście ACK/.



Rys. 6. Układ strobowanego wyjścia

W przedstawionym rozwiązaniu linie wyjściowe sterowane są tylko w okresie, gdy na linii ACK/ podawany jest poziom niski (aktywny). Odpowiada to sposobowi wyprowadzania danych przez układ 8255 przy pracy w trybie 2. Jeżeli współpraca z urządzeniem peryferyjnym wymaga sterowania linii danych przed podaniem niskiego poziomu na linię ACK/, należy dodatkowo zastosować układ generujący krótki impuls przy przejściu ACK/ z poziomu niskiego na wysoki.

Nie istnieje niestety jednolity protokół przesyłania informacji z obustronnym potwierdzeniem transmisji (ang. handshake). Dlatego też dla różnych urządzeń peryferyjnych może okazać się konieczne dodatkowe zmodyfikowanie przedstawionych propozycji.

AJP

LITERATURA

[1] 8 Bitowa Brama Wejście-Wyjście UCY 74S412. ITE; Aleja Lotników 32/46, 02-663 Warszawa
[2] Kane J., Osborne A.: An Introduction to Microcomputers, vol. 3
[3] The 8080A/9080A MOS Microprocessor Handbook. Advanced Micro Decices, 1977.

Komputerowe „zrób sam" w RFN

W RFN oferowanych jest kilka różnych mikrokomputerów, które można zakupić w częściach, a następnie samemu złożyć i uruchomić. Każdy z tych komputerów pomyślany jest jako zestaw płytek, o różnorodnych funkcjach. Cechą wspólną poniższych propozycji jest ten sam standard magistrali (ECB). Nie wiemy jednak, czy pakiety z różnych zestawów będą ze sobą współpracować.

● **mc — CP/M — Computer**

(ceny w markach; p — płytka, i — instrukcja, k — zestaw do montażu, g — zmontowany i uruchomiony pakiet)

SYS1 zawiera CPU, 64K RAM, 4K EPROM, bootstrap dla systemu operacyjnego CP/M; p+i — 69.—; k — 398.—; g — 498.—; EPROM (zawiera MONITOR) — 39.—

FLO1 sterownik do napędów dysków elastycznych 5.25" (podwójna gęstość) i 8" (pojedyncza gęstość); p+i — 69.—; k — 398.—; g—498.—; dyskietka zawierająca system operacyjny CP/M — 398.—

OUT1 zestaw szeregowych i równoległych wejść—wyjść (2 × V24 i 20 linii TTL); p+i — 65.—; k — 293.—; g — 398.—

TERM1 pakiet sprzęgu graficznego umożliwiający rozdzielczość 256 × 512 punktów, wykorzystujący dwa procesory (Z80 i GDP 9366); i — 20.—; p + 8 K ROM — 149.—; k — 698.—; g — 950.—

W numerze 4/84 pisma „mc" opublikowano sposób modyfikacji FLO1, pozwalający współpracować z napędami 8" przy wykorzystaniu podwójnej gęstości zapisu.

● **NDR — Klein — Computer** jest podstawą kursu prowadzonego przez stację telewizyjną NDR (oprócz podręcznika-instrukcji montażu można nabyć dwie wideo-kasety z nagranym kursem telewizyjnym — pierwsza cena dotyczy zestawu do montażu, druga uruchomionego pakietu; można też nabywać osobno płytki drukowane w cenie 15 marek)

SBC2 Z80A CPU, RAM, gniazdo pod EPROM — 79.95/129.— [M, 1]

IOE równoległe we-wy 2 × 8 bitów — 39.95/69.— [1]

KEY sprzęg równoległy do klawiatury — 49.95/89.— [1]

GDP 64K sprzęg graficzny z wykorzystaniem GDP 9366 — 269.—/378.— [1]

CAS sprzęg z magnetofonem kasetowym — 74.90/129.— [1]

POW5V zasilacz +5V/3A — 39.95/58.50.— [M, 1]

CPU68K pakiet procesora 68008 — 199.—/265.— [2]

ROA64 pakiet pamięci 8K × 8 RAM/ROM — 39.95/84.— [2]

PROM programator pamięci EPROM — 79.95/129.— [2]

BUS2I magistrala systemowa ECB dla sześciu pakietów — 39.95/58.50 [1]

MINIBUS magistrala dla POW5V, SBC2, IOE — 19.50/24.50 [M]

IOE-EX pakiet we—wy dla sterowania robota, generowania muzyki i in. — 89.90/145.— [M]

TAST klawiatura (montowana fabrycznie) — 198.— [1]

Oprogramowanie:

MON1 MONITOR dla SBC2 — 44.50 [1]
BASIC dla SBC2 — 50.—
MON68K MONITOR/ASSEMBLER dla CPU68K — 155.— [2]
PASCAL dla CPU68K — 155.—

Oferowane są również zestawy:

PAK-M zestaw eksperymentalny z pakietów oznaczonych [M] — 219.—/333.—
PAK-1 pakiety i oprogramowanie oznaczone [1] — 799.—/1095.—
PAK-2 pakiety i oprogramowanie oznaczone [2] — 555.—/699.—
PAK-3 pakiety i oprogramowanie oznaczone [1] lub [2] — 1349.—/1794.—

TERESA WILCZEK
Centrum Obliczeniowe SGGW-AR
Warszawa

Roboty (2)

Urządzenia kroczące

„Inwazja" robotów przemysłowych (manipulatorów) zainspirowała rozwój robotów kroczących. Już na początku ubiegłego wieku zwrócono uwagę na zalety tzw. lokomocji nożnej, umożliwiającej ruch po dowolnej nawierzchni, pokonywanie napotkanych przeszkód oraz charakteryzującej się — jak wykazały badania — zadziwiającą ekonomicznością wydatków energetycznych. Zastosowanie komputerów umożliwiło przejście od czysto mechanicznych maszyn kroczących do robotów kroczących.

Pojawiły się jednak problemy wynikające z konieczności sterowania ruchem w czasie rzeczywistym. Okazało się — na przykład — że zrealizowany za pomocą szybkiego komputera i oparty na koncepcji procesów współbieżnych układ sterowania jest zbyt wolny, ponieważ uzyskana prędkość robota kroczącego eliminuje go z jakichkolwiek zastosowań praktycznych. Z tego też powodu dalsze prace są prowadzone dwutorowo. Z jednej strony — sięga się do wyrafinowanych technik informatycznych, rozwijając koncepcje oprogramowania procesów współbieżnych, stosując systemy wieloprocesorowe, metody sztucznej inteligencji oraz metody symulacyjne; z drugiej zaś — opracowywane są różne warianty realizacji ruchu robota — tak, by sterowanie było najprostsze.

Poniżej zostanie dokonany przegląd różnych koncepcji sterowania robotami kroczącymi. Przedstawione przykłady obrazują sposoby naśladowania — za pomocą komputerowych sytemów sterowania — organizmów żywych, zarówno pod względem możliwości, jak również efektywności i szybkości działania. Należy jednak stwierdzić, że nie stworzono jeszcze dotąd zadowalająco szybkiego i sprawnego robota kroczącego. Problem jest nadal otwarty.

*

Teoria „adaptacyjnych maszyn kroczących" [9] — związana z rozwojem techniki mikroprocesorowej — powstała w ostatnim dziesięcioleciu. Urządzenia kroczące konstruowane wcześniej były sterowane wyłącznie mechanicznie.

Już podczas II wojny światowej usiłowano zastosować mechanizm kroczący w czołgach, ale pomysł ten odrzucono ze względu na trudności związane ze stabilizacją chodu [10].

Skonstruowany pod koniec lat sześćdziesiątych przez firmy BUCYRAS-ERIE COMPANY Big Muskie jest największą na świecie maszyną kroczącą [4] ze sterowaniem wyłącznie mechanicznym. Czworonożny „Muskie" waży ok. 13 tys. ton i napędzany jest przez 24 elektryczne silniki o mocy 600 KM każdy. Ruch maszyny polega na cyklicznych przemieszczeniach kończyn krokami o długości ok. 4,5 m.

Klasycznym przykładem maszyny kroczącej jest mechaniczny koń — **CAMS** (Cybernetic Anthropomorphic Machine System) zbudowany w 1964 roku przez amerykańską firmę GENERAL-ELECTRIC [6]. Masa tej również czworonożnej maszyny wynosi 1350 kg, długość — 3,5 m, a wysokość — 3,3 m. Maszyna może rozwijać prędkość do 8 km/h na terenie płaskim i ma udźwig 225 kg. Operator, znajdujący się w kabinie tej kroczącej ciężarówki, steruje ruchem jej nóg. Urządzenie może balansować, obracać się, omijać oraz pokonywać przeszkody o wysokości nie przekraczającej 1,2 m. Sterowanie takiego systemu jest zadaniem trudnym — tym bardziej, że operator znajduje się kilka metrów nad powierzchnią ziemi. Mechaniczny koń był pomyślany jako narzędzie zwiększające siłę i możliwości działania operatora.

Do celów doświadczalnych, jak choćby analiza i identyfikacja istoty chodu, budowano bardzo specyficzne mechanizmy kroczące — takie jak np. jednonożna maszyna skacząca czy różne modele maszyn dwunożnych, tzw. bipedów.

*

Pojawienie się mikroprocesorów stanowiło moment przełomowy w rozwoju teorii maszyn kroczących. Współpraca układu mechanicznego z komputerem pozwoliła teraz osiągnąć odpowiednią koordynację ruchu nóg oraz zwiększyć możliwości operacyjne tych urządzeń — doprowadzając do powstania robotów kroczących.

Pierwszym takim urządzeniem, sterowanym jeszcze tylko częściowo za pomocą komputera, był **Phoney Pony,** skonstruowany w 1966 roku w Uniwersity of Southern — California [4]. Ta czworonożna maszyna waży ok. 45 kg i ma wielkość małego kucyka. Operator obserwuje ją na ekranie monitora, a ruch odbywa się w efekcie współpracy operatora ze zrealizowanym programowo generatorem rozkazów kinematycznych.

Na Uniwersytecie Waseda w Tokio zbudowano całą serię sterowanych komputerem robotów dwunożnych. **WABOT** (Waseda Robot), posiadający nogi, ręce, uszy i głos, jest najciekawszym przykładem robota kroczącego o dużym stopniu antropomorfizmu — zarówno pod względem funkcji, jak i kształtu [6, 7]. Odpowiada on wyobrażeniu o robotach, tworzonemu w literaturze fantastyczno-naukowej. Model ten waży 130 kg i ma udźwig 15 kg. Część mechaniczno-lokomocyjna współpracuje z 16-bitowym minikomputerem HITAC-10, wyposażonym w pamięć o pojemności 16 K bajtów i programowanym w języku ASSEMBLER. Analogowo-cyfrowe urządzenia wejścia-wyjścia minikomputera łączą go z układem mechanicznym. System sterowania realizuje następujące funkcje:
— przyjmowanie i analizowanie sygnałów informujących o aktualnym położeniu nogi (dane dla tzw. generatora ruchu)
— generowanie parametrów ruchu przez generator ruchu.
Rozkazy z generatora ruchu, po odpowiedniej konwersji cyfrowo-analogowej, sterują elektro-hydraulicznymi serwomechanizmami powodującymi ruch części mechanicznej. Synchronizację ruchu umożliwia sprzężenie zwrotne zrealizowane w części mechanicznej za pomocą czujników. Sygnały generowane przez czujniki, po konwersji analogowo-cyfrowej, informują programy sterujące o pewnych, charakterystycznych położeniach nóg. Ze względu na przyjętą metodę sterowania i czas potrzebny na dokonanie programowej predykcji sygnałów sterujących, cykl kroku wynosi ok. 15 s, a krok ma długość zaledwie 15 cm. Interesujące są układy służące do komunikacji z człowiekiem, a mianowicie „uszy" i „głos" robota. Program o długości 75 K słów napisany w języku FORTRAN na komputer IBM 1800 stanowi procesor mowy.

WABOT rozpoznaje zdefiniowaną grupę rozkazów, rozpoznaje zdania zbudowane ze znanych mu słów oraz dokonuje syntezy mowy. Procesor mowy umożliwia wzbogacanie słownika w rezultacie procesu nauczania polegającego na wielokrotnym powtarzaniu słów. W procesorze mowy zastosowano metodę korekcji błędu. Efektywność nauczania jest duża, np. po 200 cyklach nauczania (powtórzeniach) robot dysponuje 13 słowami o poprawności rozpoznania na poziomie ponad 90%.

Robot sześcionożny — **HEXAPOD**, skonstruowany w Ohio State University, jest przykładem rozwiązania o szczególnie dużych możliwościach operacyjnych [3, 4]. Ruch i położenie każdego z trzech stawów, w każdej z nóg, są kontrolowane przez system komputerowy. Człowiek określa prędkość i kierunek ruchu, przy czym informacje te są automatycznie syntezowane i określają zachowanie robota. Sterowanie komputerowe pozwala na stabilizację ruchu, automatyczną adaptację chodu do nierówności terenu oraz taką optymalizację ruchu nóg, aby koszty energetyczne tego ruchu były minimalne. Robot ma długość 1,3 m, wysokość 1,4 m i waży 103 kg. Połączenie minikomputerów PDP-11/03 i PDP-11/45 tworzy interakcyjny system sterowania w czasie rzeczywistym. System ten zrealizowany w 1979 roku, jest rozwinięciem opracowanego w latach 1976—1977 sterowania jednoprocesorowego, opartego na PDP-11/45. Programy rozwiązujące równania ruchu napisane są w FORTRANIE, natomiast programy obsługi przerwań i kontroli przesyłania danych — w ASSEMBLERZE.

System ten jest pierwszym i — jak do tej pory — jedynym znanym powszechnie programowanym systemem sterowania o dużej elastyczności i efektywności, dlatego też warto podać kilka jego cech charakterystycznych. Planowanie ruchu rozwiązywane jest przez PDP-11/03, natomiast sposób realizacji zaplanowanego ruchu opracowuje PDP-11/45. Program obsługi przerwań co 1/60 sekundy przerywa obliczenia równań różnicowych ruchu (programu realizacji ruchu) i — przyjmując dane z części mechanicznej — weryfikuje pozycję nóg. Bardzo ważny jest tu czas realizacji obliczeń, ponieważ może on istotnie ograniczyć prędkość robota. W przedstawionym układzie, na skutek oddzielenia zadania planowania ruchu (PDP-11/03) od obliczeń realizacji ruchu (PDP-11/45), zwiększono maksymalną prędkość o 30%. Ulepszenie koncepcji sterowania zmniejszyło przy tym wydatki energetyczne ruchu o ponad 20%. Przy prędkości 10 cm/s pobór mocy przy sterowaniu jednoprocesorowym wynosił 2,1 KW, a przy dwuprocesorowym — 1,4 KW.

System sterowania zajmuje 8 K słów pamięci PDP-11/03 oraz 9 K słów pamięci PDP-11/45. Komunikacja między procesorami odbywa się z szybkością 6 K słów/s. Wykonanie jednego kroku programu planowania ruchu trwa 0,079—0,083/s, a wyliczenie pozycji jednej nogi — 0,025 s, przy czym oba procesy przebiegają równolegle. Jednym z kryteriów przyjętej struktury oprogramowania było równomierne obciążenie komputerów, czego jednak nie udało się w pełni zrealizować.

*

Omówione systemy sterowania mają umożliwić realizację autonomicznego robota, wyposażonego w tzw. komputer pokładowy. Uczeni radzieccy natomiast stosują łączność przewodową między maszynami kroczącymi a dużymi komputerami [1, 2, 8]. W tym przypadku badania ukierunkowane są na rozwiązanie złożonych problemów dynamiki ruchu warunkujące przystosowanie robotów kroczących do ruchu na terenach o różnym ukształtowaniu. W Instytucie Mechaniki w Moskwie prowadzone są badania nad maszynami sześcionożnymi. Zbudowano tam kilka modeli takich robotów, wykorzystując szeroko metody symulacji komputerowej. Badania symulacyjne polegały na oprogramowaniu modelu części mechanicznej robota oraz jego otoczenia (efekty symulacji wyświetlano na ekranie monitora). Drugi blok programów stanowił system sterowania ruchem. Zadając odpowiednie charakterystyki otoczenia robota (kształt terenu) badano jakość systemu sterowania. Sterowanie zmieniało model symulacyjny, a efekty zmian były widoczne jako odpowiednie zmiany monitorowego obrazu ruchu. W warunkach rzeczywistych wyposażono robota w lampę skanującą (dalmierz) informującą komputer sterujący o konfiguracji terenu. Sterowanie ruchem robota odbywa się za pomocą programów opracowanych metodami modelowania matematycznego.

Należy dodać, że prace nad robotami kroczącymi rozpoczęto również w kraju. Zespół Biomechaniki Technicznej Instytutu Techniki Lotniczej i Mechaniki Stosowanej Politechniki Warszawskiej zajmuje się zagadnieniem lokomocji czteronożnej. Aktualnie projektowany jest tam system sterowania ruchem w układzie otwartym, tzn. system umożliwiający realizację zadanej strategii ruchu bez adaptacji do kształtu nawierzchni i bez korekcji pozycji nóg.

W USA prowadzone są prace nad robotami kroczącymi do badań kosmosu [4]. Firma SPACE GENERAL CORPORATION buduje przenzaczone do tych celów maszyny sześcio- i ośmionożne. Na uwagę zasługuje tu szczególnie ośmionożny **księżycowy piechur** (ang. lunar walker). Robot ma po dwie nogi w każdym rogu podwozia, dzięki czemu w każdej fazie chodu jest podparty w czterech punktach, co znacznie zwiększa stabilność chodu. Maszyna ma sterowanie automatyczne. Ze względu na łatwość sterowania Uniwersytet Kalifornijski zaadaptował tego robota dla dzieci kalekich.

* * *

Tak pokrótce wygląda przegląd ciekawszych rozwiązań oraz ośrodków zajmujących się problematyką robotów kroczących. Prace dotyczące lokmocji nożnej (ang. legged locomotion) prowadzone są bardzo intensywnie i fakty aktualne szybko stają się historią. Ze względu jednak na dużą przydatność robotów kroczących do celów militarnych, nie wszystkie wyniki badań są publikowane.

Symulacja komputerowa ruchu, metody rozpoznawania obrazów, metody programowania procesów współbieżnych w czasie rzeczywistym, a także komputerowa analiza mowy — stanowią podstawę prac badawczych w dziedzinie robotów kroczących, które obecnie programuje się w oparciu o powszechnie stosowane języki. Można jednak przewidywać, że rozwój tej dziedziny spowoduje wkrótce powstanie języków specjalizowanych, podobnie jak w przypadku sterowania manipulatorami robotów przemysłowych.

LITERATURA

[1] Aristova M. V., Ignatiev M. B., Prokhorov V. M.: Algorythmic system for robot's motion simulation. 2nd Symposium on Theory and Practice of Robots and Manipulators. PWN, Warszawa, 1977

[2] Bessonov A. D., Umnov N. V.: Features of kinematics of turn of walking vehicles. 3rd Symposium on Theory and Praktice of Robots and Manipulators. PWN, Warszawa, 1980

[3] Ching-Shu Chac M. S.: Real-time multiprocessor control of a hexapod vehicle. Dissertation. The Ohio State University, 1979

[4] Jaswa V. C.: An experimental study of real-time computer control of a hexapod vehicle. Dissertation. The Ohio State University, 1978

[5] McGhee R. B., Chao C. S., Jaswa V. C., Orin O. E.: Real-time computer control of a hexapod vehicle. 3rd Symposium on Theory and Praktice of Robots and Manipulators. PWN, Warszawa, 1980

[6] Morecki A., Ekiel J., Fidelus K.: Cybernetyczne systemy ruchu kończyn zwierząt i robotów. PWN, Warszawa, 1979

[7] Ogo K., Ganse A., Kato I.: Quasi dynamic walking of biped walking machine aiming at completion of steady walking. 3rd Symposium on Theory and Praktice of Robots and Manipulators. PWN, Warszawa, 1978

[8] Ochocimockij D. E., Płatonov A. K., Gurfinkiel W. S., Déanin E. A.: Razrabotka integralnovo szagajuszczevo robota. Ustrojczivost dviżenija. Analiticzeskaja mechanika. Upravlenije dwiżenijem. Nauka, Moskwa, 1981

[9] Simons G. L.: Robots in Industry. NCC, Oxford, 1980

[10] Thring M. W.: Some experimentor walking machines. 3 rd Symposium on Theory and Praktice of Robots and Manipulators. PWN, Warszawa, 1980

[11] Young J. F.: Robotics. Buttenworths, London, 1973.

Obudowa

Rozmowa z Ryszardem Kajkowskim, szefem firmy CSK, produkującej komputery i oprogramowanie użytkowe

Redakcja: — Jest Pan informatykiem z otwartym przewodem doktorskim, czyli planował Pan karierę naukową. Jednak Pana nazwisko znane jest z działalności firmy, która zdobyła rozgłos produkując LIDIĘ (komputer kompatybilny z APPLE II). O sukcesie świadczy fakt, że zamówienia przekroczyły zdolności produkcyjne CSK na najbliższe trzy lata. Jakby tego nie było dosyć, rozszerza Pan ofertę o oprogramowanie użytkowe dla mikrokomputerów, którego nikt dotąd w Polsce nie sprzedawał. Zaczynał Pan jednak od zera. Co skłoniło Pana do wyboru takiej właśnie drogi życiowej?

Ryszard Kajkowski: — W pewnym momencie doszedłem do wniosku, że aby się rozwijać sam muszę o to zadbać. A informatyka to chyba najtańsza branża.

Red.: — Ze względu na inwestycje?

R.K.: — Tak! W 1980 roku udało mi się tanio kupić mikrokomputer APPLE. Zacząłem więc zastanawiać się co z tym można zrobić. To był początek.

Red.: — A nie próbował Pan działać w przedsiębiorstwach państwowych?

R.K.: — Próbowałem. Przez trzy lata pracowałem w dużym ośrodku obliczeniowym jako konstruktor systemów. Odszedłem, gdy zmierzch informatyki w przedsiębiorstwach państwowych stał się całkiem widoczny. Nawet nie ze względu na tematykę, którą się zajmują. Niestety, specyfika tych przedsiębiorstw jest taka, że inicjatywa jest tłamszona — po prostu nie ma możliwości rozwoju.

Red.: — Zaczął Pan więc jako „inicjatywa prywatna"?

R.K.: — Po wielu kłopotach dostałem zezwolenie z Ministerstwa Handlu Wewnętrznego i Usług na prowadzenie działalności usługowej związanej z oprogramowaniem. Ponieważ dostałem zamówienie z RFN na wykonanie oprogramowania dla mikrokomputera, zależało mi na stworzeniu podstaw formalnych. W trakcie realizacji umowy zorientowałem się, że ze względu na rodzimy niedorozwój mikroinformatyki, zasadniczym problemem jest sprzęt, a nie oprogramowanie. Nie miałem więc innego wyjścia — musiałem sam sobie stworzyć narzędzia, aby móc w przyszłości zabrać się za oprogramowanie. Założyłem firmę rzemieślniczą, która zajęła się produkcją mikrokomputerów.

Red.: — Wielu informatyków podziela Pana frustrację związaną z pracą w przedsiębiorstwach państwowych. Większość jednak boi się ryzyka. Jaki jest naprawdę stosunek urzędów do inicjatyw takich jak Pańska?

R.K.: — Natknąłem się na bardzo skrajne przypadki. Na poziomie najniższym udało mi się sprawę przepchnąć chyba tylko dzięki podejściu Urzędu Miejskiego w Tczewie. Na szczeblu wojewódzkim byłem natomiast traktowany z bardzo dużą rezerwą — sprawa nie mieściła się w ich pojęciach, nomenklaturach, tabelach usług... Na poziomie najwyższym, MHWiU — bez specjalnych priorytetów, ale i bez zahamowań. Wszystko w normalnym trybie, z zachowaniem regulaminowych terminów. Nawet z dyskretną ciekawością — co z tego będzie. W Rzemiośle miałem już drogę przetartą — dysponowałem gotowym produktem, no i realizowałem jakieś formy eksportu.

Red.: — Krajowa oferta w dziedzinie mikrokomputerów pochodzi głównie od firm prywatnych (polonijnych lub rzemieślniczych). Popyt na ich produkty poważnie przewyższa podaż. Dlaczego firmy prywatne, z natury przecież bardzo elastyczne, nie zwiększą produkcji — aby ten popyt zaspokoić?

R.K.: — Podstawowym problemem w produkcji mikrokomputerów są... urządzenia peryferyjne. To blokuje moje przedsięwzięcia, jak i zapewne firm polonijnych. Na polskim rynku praktycznie nie ma dostępu do napędów dyskietek, monitorów, klawiatur i drukarek. Producent drukarek nawet nie chce rozpocząć rozmów przed rokiem 1985, MERA-KFAP (producent napędów do dyskietek) w ogóle nie reaguje na jakiekolwiek zapytania...

Red.: — Ale ten sprzęt jest przecież produkowany i w innych krajach RWPG: NRD, Węgry, Bułgaria...

R.K.: — Właśnie. Lecz przy centralnym rozdysponowaniu urządzeń z importu firmy prywatne nie są w zasadzie brane pod uwagę. Nie to jednak jest podstawowym błędem central zajmujących się importem. Otóż analizują one popyt, który przy braku podaży praktycznie się nie objawia. Jeśli pytanie o potrzeby kierowane jest do dużego producenta, to odpowiedź jest na miarę możliwości przedsiębiorstwa. Import jest więc niewielki — i tak zamyka się krąg absurdu.

Red.: — Być może ten absurd ma tu swoje głębsze podłoże?

R.K.: — Nie ma niestety społecznej świadomości problemu mikroinformatyki w Polsce. Na świecie 80% mikrokomputerów kupowanych jest przez odbiorców spoza kręgów dużych firm: osoby prywatne, sklepy, szkoły, uczelnie...

Red.: — Jest to więc towar rynkowy, mogący w naszej sytuacji ściągnąć pokaźną część „jałowych" pieniędzy.

R.K.: — To zrozumiałe. Obecnie cena jest niewspółmierna do ponoszonych kosztów, a można sądzić, że u nas się ona utrzyma. Popyt jest już spory — szczególnie młode pokolenie wykazuje zainteresowanie.

Red.: — Produkcja państwowych potentatów rozbija się zapewne o brak stosunkowo niewielkiej ilości dewiz, które nie zwrócą się z eksportu mikrokomputerów — zanadto jesteśmy w tej dziedzinie opóźnieni.

R.K.: — Chyba raczej o brak elastyczności w myśleniu. Decyzje są na razie w ręku „sprzętowców". A od nich trudno oczekiwać zrozumienia, że na informatyce można zarabiać dewizy robiąc na przykład... pudełka do dyskietek. Proponowałem przedsiębiorstwom państwowym: zróbcie obudowy do komputerów i monitorów, a dostaniecie za to mikroprocesory. Za obudowę dostaniecie mikrokomputer! Prosta prawda jest jednak poza granicami percepcji decydentów. Jak poważne przedsiębiorstwo elektroniczne, stworzone do Wielkiej Informatyki, mogłoby się zajmować banalną kwestią obudowy?! Produkcja mikrokomputerów nie może być realizowana przez hobbystów i studentów — to jest cały przemysł, w którym nie należy kierować się ambicją, lecz rozeznaniem i zdrowym rozsądkiem. Sprzedać na Zachodzie komputer jest bardzo trudno; wejść w kooperację — znacznie łatwiej.

Red.: — Może jednak odpowiednio szeroki ruch hobbystyczny miałby szansę wywarcia presji, która zmusiłaby decydentów do konkretnych posunięć?

R.K.: — To bardzo trudne, gdyż w naszym kraju tzw. nacisk oddolny musi mieć instytucjonalne formy. Musi wykorzystywać prasę albo jakieś organizacje społeczne związane z informatyką. Poza tym, naciski nie docierają do decydentów w odpowiednio ostrej formie i rodzi to błędy na etapie zarządzania. Błędy najgorsze w skutkach. Chyba warto by rozbudować szkolenia dające szansę przekazania tzw. świadomości informatycznej. Może decydenci zrozumieliby wreszcie, że o upowszechnienie mikrokomputera powinni zabiegać także we własnym interesie.

Red.: — Mogą też obawiać się nowej techniki, która bezpardonowo ujawnia brak kompetencji.

R.K.: — W przypadku mikrokomputera możliwe jest wyeliminowanie pośrednictwa programistów. Wykorzystywanie programów użytkowych nie musi wymagać przygotowania informatycznego (bardzo ważny aspekt psychologiczny!). Oczywiście, starsze pokolenie ze swoimi przyzwyczajeniami będzie tak czy inaczej niechętne wszelkim innowacjom. Ale młodzi, wyposażeni w nowe narzędzia, doprowadzą w łagodnej, a może i drastycznej formie — do przyspieszenia procesu wymiany kadr.

Red.: — Te narzędzia muszą się jednak skądś wziąć.

R.K.: — Tak, lecz problem mikroinformatyki w Polsce nie ogranicza się do sprzętu. To również kwestia prawna, stosowanie udogodnień finansowych, kredytów. Dopiero całościowe ujęcie tego problemu może dać jakieś satysfakcjonujące efekty. Na całym świecie mikroelektronika jest forsowana odpowiednimi programami rządowymi.

Red.: — No tak, ale u nas władza jakoś nie jest zainteresowana forsowaniem mikroinformatyki.

R.K.: — Niezupełnie. Choćby niektóre prace w Instytucie Podstaw Informatyki prowadzone są w oparciu o zlecenia rządowe. Lecz problem w tym, że świadomość decydentów ogranicza się w zasadzie do dużych maszyn. Nie ma jeszcze kadry, która poznałaby fenomen mikrokomputera. Dlatego też podejmowane są decyzje, które nie pasują do dzisiejszej mikroelektroniki. Przykładem mogą być programy związane z elektronizacją medycyny. Opiera się ona na sprzęcie specjalizowanym, na który nakłada się szczególnie wysokie wymagania — sprzęcie produkowanym jednostkowo. Idea pożyteczna, tyle że elektronika masowa nie może mieć podstawy w zastosowaniach specjalistycznych, lecz właśnie odwrotnie — rozwój elektroniki masowej może stworzyć warunki zastosowania jej w specyficznych dziedzinach.

Red.: — Skoro mowa o masowości: pokutuje u nas mit polskiego mikroprocesora. Mit — bo produkowany jest on w ilościach dalekich od masowości.

R.K.: — Na sukces firmy APPLE pracował praktycznie cały świat i to zarówno w dziedzinie sprzętu, jak i oprogramowania. U nas także muszą zapaść decyzje na odpowiednio wysokim szczeblu, które umożliwią dopływ niezbędnych podzespołów.

Red.: — Podobno nie mamy za co kupować...

R.K.: — Ale też nie sposób oprzeć gospodarki wyłącznie na kopalinach. Abyśmy mogli kupować, musimy zacząć produkować nowocześniej. A tego nie da się robić bez mikrokomputerów. W interesie wszystkich gałęzi przemysłu leży inwestowanie w mikroelektronikę; tylko wtedy będziemy w stanie sprzedawać nasze produkty w jakiejś rozsądnej kalkulacji. W tej chwili jesteśmy za biedni, żeby popełniać błędy. To już nie jest problem — czy nas stać czy nie. Nas musi być na to stać!

Red.: — Inaczej grozi nam dalsze oddalenie się od świata...

R.K.: — I to w szybkim tempie. Obecnie postęp techniczny widać na przestrzeni kilku miesięcy. Po prostu komputer robi komputer — to jest przyspieszenie w postępie geometrycznym. Z tego trzeba zdać sobie sprawę. Nie mamy innego wyboru.

Rozmawiał:
ANDRZEJ J. PIOTROWSKI

O mikroprocesorach po polsku

Kolejna konferencja dotycząca mikroprocesorów [1], pn. „Mikroprocesory — stan i perspektywy zastosowań w Polsce", odbyła się w dn. 18—19 października ub.r. w Kołobrzegu, zorganizowana przez Oddział Wielkopolski Towarzystwa Naukowego Organizacji i Kierowania, przy współudziale Komitetu ds. Informatyki Oddziału Wojewódzkiego NOT w Poznaniu. Przedstawiono na niej 13 referatów problemowych i kilkanaście komunikatów, w których poruszono wiele zagadnień istotnych dla rozwoju techniki mikroprocesorowej i jej zastosowań. Obrady toczyły się równolegle w dwóch sekcjach tematycznych:

- sprzęt, oprogramowanie, systemy
- zastosowania.

Poniżej zmodyfikuję nieco ten podział, wyróżniając dodatkowo grupę referatów o znaczeniu bardziej uniwersalnym.

SYSTEMY

Starałem się zwrócić uwagę przede wszystkim na opracowania konstrukcyjne, lecz nowych opracowań było bardzo mało. Jedynym godnym odnotowania, bo dotąd mi nie znanym, był — oparty na mikrokomputerze jednopłytowym — sterownik zrealizowany w Instytucie Technologii Elektronicznej Politechniki Gdańskiej, przedstawiony w dwóch odmianach — uniwersalnej, przeznaczonej do zastosowań kontrolno-pomiarowych (J. Gajkiewicz i in.), i specjalizowanej — jako tester układów scalonych (A. J. Majewski). Inne prezentowane konstrukcje (sterowniki lub zestawy) są na ogół znane naszym czytelnikom (por. np. K. Rzymkowski, nr 2, 1983). W tej dziedzinie nie zanotowano więc znaczącego postępu, np. w porównaniu z rozwiązaniami przedstawionymi w ubiegłorocznych, „mikroprocesorowych" numerach INFORMATYKI.

Pewną nowością jest pojawienie się prac dotyczących systemów wieloprocesorowych, jak np. magistrali systemu wieloprocesorowego (M. Domżalski, Instytut Informatyki PŁ) lub jego struktury (J. Jaworowski, J. Zaczek, Instytut Informatyki AGH). Są to jednak prace typowo badawcze, a ich efekty będą zależeć — moim zdaniem — w znacznym stopniu od umiejętnego dostosowania się do standardów światowych w tej dziedzinie.

W rozwoju oprogramowania krajowych mikrokomputerów nadal nie mogę dopatrzyć się śladu generalnej koncepcji, a nie chcę uwierzyć w to, że takowej nam nie potrzeba. Najciekawszym, z tego punktu widzenia, zjawiskiem omawianej konferencji były prace nad oprogramowaniem dla mikrokomputera PSPD-90, który jest pewnym ewenementem na naszym rynku; powstał bowiem z programowanej stacji przetwarzania danych, produkowanej przez krakowski KFAP (w oparciu o mikroprocesor 8080). Na konferencji przedstawiono — opracowany w Instytucie Informatyki UJ — system operacyjny MINOS (K. Jojczyk, M. Kubowicz) i kompilator języka PL/M (T. Kędzierski). Ambicją autorów jest stworzenie pełnego, a jednocześnie minimalnego środowiska programowego dla tego mikrokomputera. Oprócz wymienionych składników, oprogramowanie podstawowe PSPD-90 zawiera także edytor tekstowy, makroasembler, konsolidator, bibliotekarz i program uruchomieniowy (wzorowane częściowo na odpowiednich narzędziach systemu operacyjnego RT-11). Istotnym ograniczeniem przy realizacji zamierzeń i zadań systemu mogą być wrodzone wady mikrokomputera, jak niedoskonałość obsługi przerwań czy mała pojemność pamięci operacyjnej.

Znamiona produktu programowego, przystosowanego do ewentualnego rozpowszechniania, nosi tylko jeden pakiet — arytmetyka zmiennoprzecinkowa dla mikroprocesora Z-80 i — w założeniach — dla MCY7880 (A. Jędrzejewski, Centrum Uczelniano-Przemysłowe Metrologii i Systemów Pomiarowych PW). Autor nie podaje jednak, czy przyjęty przez niego sposób reprezentacji jest zgodny ze standardem międzynarodowym, np. firmy INTEL lub organizacji IEEE, co jest bardzo istotne.

Stosunkowo najwięcej prac dotyczyło zagadnień związanych z projektowaniem i uruchamianiem systemów mikroprocesorowych. Omawiano zarówno pojedyncze urządzenia służące do uruchamiania i testowania pakietów, np. oparte na metodzie analizy sygnatur (E. Michta, WSI Zielona Góra), jak i kompletne zestawy przeznaczone do uruchamiania mikrokomputerów, np. emulatory dla znanego systemu MSWP [2] (L. Naumowski, Instytut Maszyn Matematycznych). Przedstawiono także kilka prac dotyczących wspomaganej produkcji oprogramowania. Aktywność pracowników Instytutu Informatyki UJ przejawiła się ponownie w próbie wykorzystania mikrokomputera PSPD-90 do wzbogacenia środowiska programowego innych komputerów (W. Burczyk, K. Jojczyk). PSPD-90 jest od dawna wykorzystywany w najprostszy sposób jako stacja przygotowania danych, a więc w trybie rozłączonym (ang. off-line).

---

[1] Poprzednie konferencje krajowe na ten temat omówiliśmy w nr 2, 1981 i 4, 1983 INFORMATYKI

[2] Sinkiewicz T.: Mikroprocesorowy system wspomagania projektowania, INFORMATYKA, nr 1, 1983, str. 8—9

Znacznym rozszerzeniem możliwości tej stacji jest użycie jej jako inteligentnego terminala sprzężonego z komputerem linią transmisji, np. według standardu V.24. Kolejnym krokiem w kierunku wszechstronniejszego wykorzystania stacji będzie zastosowanie jej do wytwarzania oprogramowania dla komputerów osobistych, jak np. ZX-81 lub ORIC.

Właśnie w kierunku tworzenia skrośnego oprogramowania dla mikrokomputerów zmierzają prace nad systemem COSMIC, prowadzone w ośrodku poznańskim (K. Kurpiński, Środowiskowy Ośrodek Informatyki PP). W oparciu o instalację ODRY-1305 z systemem operacyjnym GEORGE-3, opracowano zespół programów (m.in. asembler, symulator, program uruchomieniowy) służących do tworzenia oprogramowania dla mikroprocesora 8080 (MCY7880N).

Znaczenie organizacji całego procesu projektowania i uruchamiania systemów mikrokomputerowych oraz rolę narzędzi w tym procesie podkreślono w dwóch innych referatach (A. Skorupski, J. Sosnowski, Instytut Informatyki PW), wyróżniając szereg faz, których właściwy przebieg jest istotny dla opracowania systemu:

— sformułowanie założeń technicznych
— projekt sprzętu i oprogramowania oraz realizacja modelu
— uruchamianie, testowanie i optymalizacja (przygotowanie prototypu).

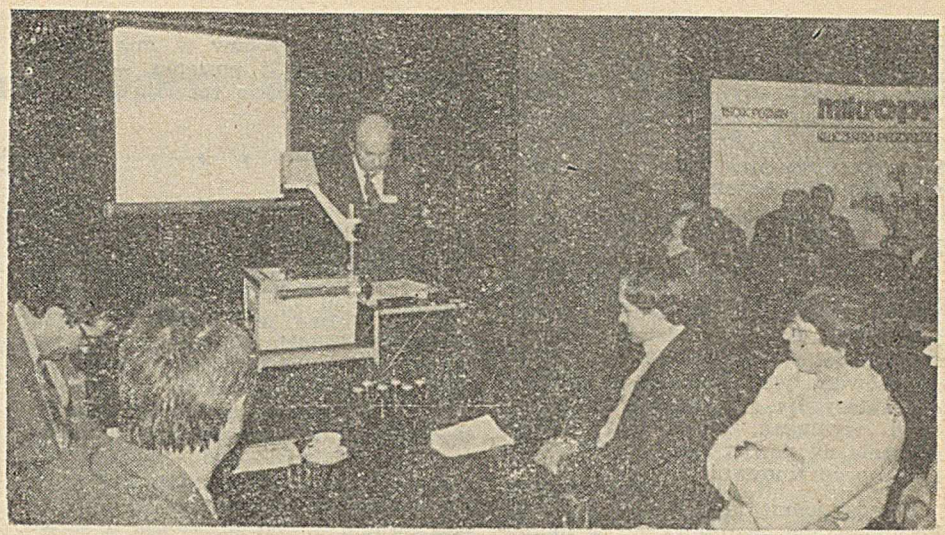
Stwierdzono, że szczególny postęp dokonuje się w sferze środków wspomagających końcowe etapy tego procesu — zwłaszcza, jeśli chodzi o urządzenia do analizy i emulacji. Choć autorzy nie przedstawili na konferencji żadnego, opracowanego przez siebie narzędzia (por. INFORMATYKA, nr 2, 1983, str. 17 i nr 6, 1984, str. 6), warto zwrócić uwagę na te referaty, gdyż są przykładem bardzo logicznego podejścia do organizacji procesu projektowania skomputeryzowanych urządzeń.

ZAGADNIENIA PODSTAWOWE

Interesującym tłem dla pojedynczych wystąpień było omówienie aktualnych tendencji światowych w dziedzinie architektury systemów komputerowych (R. Marczyński, Instytut Podstaw Informatyki PAN). Główną cechą obecnych projektów badawczych, realizowanych głównie w USA i w Japonii, obejmujących:

— język wysokiego poziomu (ang. high order language, HOL)
— bardzo szybkie układy scalone (ang. very high speed integrated circuits, VHSIC)
— systemy komputerowe piątej generacji (ang. fifth generation computer systems)
— wieloprojektowe struktury półprzewodnikowe (ang. multi-project chips, MPC),

są ogromne nakłady finansowe. Każde z tych przedsięwzięć jest zakrojone na ogromną skalę i — jak dotąd — tylko jedno z nich zostało zakończone. Ilustracji tych tendencji w skali krajowej trudno się dopatrzyć. Na przeszkodzie w realizacji takich zadań stoi chyba nie tylko brak pieniędzy — nie mogę wyobrazić sobie w kraju sprawnej organizacji dużego przedsięwzięcia z dziedziny informatyki.



Obraz konferencji

Wystąpieniem, które w dużym stopniu uzupełniało poprzednie oraz rozszerzało spojrzenie przeciętnego użytkownika na rozwój architektury systemów komputerowych, był referat dotyczący wpływu mikroprocesorów na rozprzestrzenienie mocy obliczeniowej komputerów i powstanie tzw. sieci lokalnych, ang. local area networks, LAN (A. Gościński, Instytut Informatyki AGH). Potwierdzono w nim wcześniejsze spostrzeżenia, że w Polsce istotną barierę rozwoju tej techniki będzie stopień komplikacji urządzeń sprzęgających komputery z siecią. Nie jest też nadal jasne, który z dwóch niezależnych projektów norm — IEC PROWAY czy IEEE P-802 — uzyska przewagę, choć obecnie wydaje się, że obie specyfikacje zaczynają być zbieżne.

Przegląd światowych tendencji w dziedzinie oprogramowania systemów mikrokomputerowych (J. Madey, Instytut Informatyki UW) był równie wymowny jak poprzednie, choć nie dlatego, że gdzie indziej stosuje się nowocześniejsze metody wytwarzania oprogramowania. Wystarczyło przypomnieć, że oprogramowanie powinno być: „Potrzebne (!), poprawne, tanie, łatwe w użyciu, przenośne i niezawodne". Ciekawe jest również spostrzeżenie, że przez masowość nowej technologii oprogramowanie nie jest już wytwarzane tylko przez specjalistów — piszą je coraz częściej amatorzy; i nie trafia ono do zamkniętego kręgu odbiorców — korzystają z niego coraz powszechniej nowet dzieci.

W przeciwieństwie do poprzednich konferencji dotyczących mikroprocesorów, tym razem autorzy i organizatorzy nie stronili od prac o profilu teoretycznym. Przygotowane głównie przez przedstawicieli ośrodka poznańskiego obejmowały cały zakres tematyczny konferencji — od zagadnień konfiguracji, jak np. organizacja i dobór pamięci (G. Bartoszewicz), przez podstawowe problemy programowania (J. Koperski) i generowania programów użytkowych (B. Mikołajczak) — do zagadnień integracji struktur sprzętowych i programowych w projektowaniu systemów komputerowych (Z. Liszyński). Do tej klasy prac zaliczyłbym również referaty z ośrodka krakowskiego, dotyczące języka programowania systemów wieloprocesorowych i generatora analizatorów składni języków specjalizowanych. Choć, prawdę mówiąc, trudno mi ocenić ich wartość, sądzę że wychylenie głowy poza poziom napięć TTL i pojedynczych bitów na pewno nie zaszkodziło uczestnikom konferencji.

ZASTOSOWANIA

Większość referatów dotyczących zastosowań stanowiły prace przeglądowe. W technice pomiarowej (np. A. Sowiński, Przemysłowy Instytut Elektroniki) konstruowanie przyrządów z użyciem mikroprocesorów jest już na porządku dziennym. Wbudowanie mikroprocesora do woltomierza zwiększa znacznie jego możliwości pomiarowe — tak, że oprócz napięcia, prądu i rezystancji, można nim mierzyć wzmocnienie lub tłumienie oraz wykonywać dodatkowe operacje, jak sortowanie wyników w różnych granicach, detekcję przekroczenia wartości granicznych, zapamiętywanie odczytów, skalowanie itp. Danych ilościowych o podobnym charakterze dostarczają oscyloskopy cyfrowe. Jednakże rozwój aparatury wyposażonej w mikroprocesory jest ciągle jeszcze, szczególnie w Polsce, w stadium początkowym. Z drugiej strony — wielu potencjalnych użytkowników tej aparatury nie uświadamia sobie jeszcze jej wszystkich możliwości.

W innym referacie (P. Refermat, Instytut Organizacji i Zarządzania AE w Poznaniu) omówiono zagadnienia automatyzacji biur przy użyciu mikroprocesorów, zwracając uwagę m.in. na istotne znaczenie lokalnych sieci kom-

puterowych w tych zastosowaniach. Przedstawiono również możliwości użycia mikroprocesorów w automatyce, do realizacji regulatorów cyfrowych i filtrów, analizując zalety tej techniki dla konkretnych typów mikroprocesorów (A. Kasiński, W. Bajsert, Instytut Automatyki PP).

Po raz kolejny odnotowałem z żalem brak referatu (choć gp zapowiadano) na temat zastosowania mikroprocesorów w telekomunikacji. Byłoby to interesujące — nie tylko dlatego, że warto coś wiedzieć o rozwoju tej dziedziny w Polsce. Ten rodzaj zastosowań jest dość specyficzny i charakteryzuje się istnieniem specjalnych wymagań, wpływających dość znacznie na pewną nietypowość sprzętu i oprogramowania.

Przedstawiono natomiast inne, rzadko omawiane zastosowanie, a mianowicie — wykorzystanie mikroprocesorów w procesie dydaktycznym (F. Wagner i in., Wyższa Szkoła Inżynierska, Zielona Góra). Laboratorium mikrokomputerowe, zrealizowane w oparciu o cztery kasety sterowane mikroprocesorami 8080A lub Z-80, połączone z minikomputerem SM-3, stanowi dostatecznie złożony, lecz funkcjonalny zestaw, na którym można realizować wiele ćwiczeń w nauczaniu techniki mikroprocesorowej. Ci sami autorzy przedstawili (jedyną, na tej konferencji — jak mi się wydaje) realizację urządzenia z wybudowanym mikroprocesorem — sterowanie pisakiem KL-2. Właśnie tego rodzaju zastosowania, jak wyposażenie wyrobów w mikroprocesory, świadczą o postępie — i one powinny być treścią rozwoju tej techniki w Polsce.

* * *

Co można powiedzieć — na podstawie przebiegu obrad — o stanie i perspektywach tej techniki w kraju? Organizatorzy uważają, że „dotychczasowy rozwój techniki mikroprocesorowej w Polsce jest powolny, chociaż w wielu dziedzinach wyraźny i — co ważne — systematyczny... Istnieje pilna potrzeba opracowania zunifikowanej bazy konstrukcyjnej do budowy systemów mikroprocesorowych, odpowiadających sobie pod względem mechanicznym, elektrycznym i logicznym. Konieczne jest skrócenie cyklu projektowania i wykonywania urządzeń, zwiększenia ich niezawodności oraz zapewnienia właściwych warunków konserwacji. Stworzyłoby to warunki do budowy bardziej złożonych systemów wielomikroprocesorowych czy sieci mikrokomputerowych. Konieczne jest również zwiększenie nakładów na oprogramowanie (...)." Lista tych „konieczności" mogłaby być długa, o czym pisałem w sprawozdaniach z poprzednich konferencji — nie ma więc sensu ich powtarzać [3].

[3] Por. także: J. Zalewski, W przemyśle bez zmian, Biuletyn Techniczno-Informacyjny MERA, nr 1, 1984

Warto jednakże przytoczyć głosy niektórych dyskutantów, zwracających uwagę na kwestie ogólniejsze, które mogą stać się hamulcem postępu, jak np.:

— brak mikroprocesorów 16-bitowych i segmentowych, co powstrzymuje rozwój zastosowań w telekomunikacji i informatyce

— zagrożenie monokulturą mikroprocesorową, choć wielu konstruktorów cieszy się z tego, że ten polski mikroprocesor wreszcie jest

— dylemat, czego należy uczyć w szkołach i uczelniach: czy tego, co jest obecnie konsumowane czy tego, co ma przyszłość

— brak normalizacji złącz, pakietów, płytek i sprzęgów oraz brak perspektyw w zakresie kierunków badawczych, co chyba najbardziej opóźnia postęp.

Reasumując — nie można udzielić wyczerpującej i ostatecznej odpowiedzi na pytanie o stan i perspektywy techniki mikroprocesorowej w Polsce, choćby ze względu na brak wystąpień producentów (niemal wszystkie nadesłane referaty pochodziły ze środowisk naukowych). Na pewno warto jednak organizować nadal podobne konferencje, choćby po to, by uświadomić sobie przynajmniej część barier na drodze, którą inni idą szybciej od nas.

JANUSZ ZALEWSKI

KSIĄŻKI NADESŁANE

WYDAWNICTWA NAUKOWO-TECHNICZNE

Gordon M. J. C.: Denotacyjny opis języków programowania. Seria BIO, 209 str., nakład 2000 egz., cena 100 zł, Warszawa, 1983 (tłum. A. Terlecki).

Książka omawia denotacyjną metodę opisu semantyki języków programowania. Przedstawiono w niej techniki wystarczające do definiowania semantyki języków klasy PASCALA. Książka uzupełniona jest o „Zarys teorii równań stałopunktowych semantyki denotacyjnej" pióra A. Blikłego.

Misiurewicz P.: Układy mikroprocesorowe. Seria: Układy i systemy elektroniczne. 324 str., nakład 20 000 egz., Warszawa, 1983.

Recenzję tej książki zamieściliśmy w nr. 6/1984.

Seidler J.: Nauka o informacji. Tom I: Podstawy, modele źródeł i wstępne przetwarzanie informacji. 435 str.; Tom II: Sygnały niosące informację i jej odtwarzanie, 530 str. Seria EIT, nakład 3000 egz., cena t. I/II 300 zł, Warszawa, 1983.

W monografii przedstawiona została problematyka przenoszenia, rejestrowania i odtwarzania informacji. Pierwsza część książki stanowi wstępny wykład całości materiału. Omówione są w niej podstawowe pojęcia i modele systemów informacyjnych, optymalizacja oraz przykłady tych systemów. W kolejnych dwóch częściach opisane zostały modele źródeł informacji ziarnistej i ciągłej (w tym miara nieokreśloności statystycznej oraz wstępne przetwarzanie informacji obu rodzajów). Czwarta część monografii dotyczy sygnałów niosących informację. Autor omawia tu m.in. kody korekcyjne, sygnały modulowane oraz kanały. Ostatnie dwie części poświęcone są problematyce odtwarzania informacji.

Brady J. M.: Informatyka teoretyczna w ujęciu programistycznym. Seria BIO, 325 str., nakład 4000 egz., cena 190 zł, Warszawa, 1983 (tłum A. Skowron i T. Kulisiewicz).

Książka stanowi wprowadzenie w te działy teoretycznych podstaw informatyki, które są bezpośrednio związane z praktyką programowania. Kolejne rozdziały zapoznają czytelnika m.in. z zagadnieniami metainformatyki, ograniczeniami współczesnych komputerów (np. z problemami nierozstrzygalnymi), ze złożonością obliczeniową, dowodzeniem poprawności programów oraz z pewnymi problemami semantyki języków programowania.

Pawlak Z.: Systemy informacyjne, podstawy teoretyczne. 186 str., nakład 5000 egz., cena 100 zł, Warszawa, 1983.

Książka zawiera formalną definicję systemu informacyjnego oraz związanego z nim języka. Badane są w niej zagadnienia dotyczące systemów wielostopniowych i hierarchicznych, systemów rozproszonych, systemów wielowartościowych, przybliżonych i stochastycznych. Przedstawiona została także przybliżona klasyfikacja obiektów i niepełna klasyfikacja obiektów.

Walasek J.: Konwersacyjne otoczenie programowe PASCALA. Seria BIO, 152 str., nakład 4000 egz., cena 80 zł, Warszawa, 1983.

Książka stanowi podręcznik użytkownika systemu konwersacyjnego wspomagającego produkcję programów PASCALOWYCH na maszynie JS lub IBM 360/370 pracujących pod systemem OS MVT (TSO). Oprócz otoczenia PASCALA czytelnik znajdzie nieco wiadomości także o otoczeniach konwersacyjnych FORTRANU i PL/I.

Zastosowanie mikroprocesorów

W dniach 18—21 października 1983 w Budapeszcie odbyło się trzecie międzynarodowe sympozjum poświęcone zastosowaniom mikroinformatyki. Wzięło w nim udział ok. 500 uczestników, w przeważającej części z krajów RWPG, a także kilkanaście osób z RFN, Wielkiej Brytanii, Austrii, Włoch, Japonii i in. Przedstawione referaty zgrupowane były w kilkanaście zespołów tematycznych: architektura mikroprocesorów, systemy rozproszone, przetwarzanie danych i słów, przetwarzanie sygnałów, sterowanie i pomiary, języki i programy, zastosowania w telekomunikacji, testowanie i inne. Poszczególnym tematom poświęcone były oddzielne sesje.

Na szczególną uwagę zasługują następujące referaty przeglądowe:

- Rynek mikroprocesorowy w Japonii — prof. K. Agusa, Uniwersytet Kioto
- Narzędzia programowe stosujące kolorowe monitory graficzne o dużej rozdzielczości — prof. A. C. Davies, Uniwersytet Londyński
- Stan obecny i kierunki rozwojowe w technologii mikrokomputerów do zastosowań przemysłowych — prof. W. Rembold, Uniwersytet Karlsruhe
- Organizacja systemów komputerowych, opartych na mikroprocesorach segmentowych — prof. L. A. Szumiłow, Instytut Elektrotechniki w Leningradzie
- Mikrokomputery o wyspecjalizowanej architekturze, dostosowanej do algorytmów przetwarzania informacji o określonej strukturze — prof. R. M. Lea, Uniwersytet Brunel (Wielka Brytania)
- Rozbieżne kierunki w rozwoju architektury mikroprocesorów — prof. L. Richter, Uniwersytet Dortmundzki.

Bardzo interesujący był również, zamieszczony w materiałach, choć nie wygłoszony, referat prof. B. Součka z Uniwersytetu w Zagrzebiu, na temat czwartej i piątej generacji oprogramowania.

W niniejszym omówieniu przedstawione zostaną jedynie tematy wiodące, poruszone przekrojowo w głównych referatach, najlepiej ilustrujące obecne tendencje światowe — bez wnikania w zagadnienia szczegółowe.

*

Główne tendencje rozwojowe w technice mikroprocesorowej są następujące:

- **Pamięci dynamiczne RAM o pojemności 16 K bitów (16 Kb DRAM) zastępowane są pamięciami 64 Kb DRAM. Rozpoczyna się wprowadzanie na rynek pamięci 256 Kb DRAM; pamięć taka zawiera ok. 550 tys. elementów, a w 1988 roku spodziewane jest pojawienie się pamięci 1 Mb DRAM, która zawierać będzie ponad 1,5 mln elementów.**
- **Spośród pamięci statycznych RAM (SRAM) najwięcej sprzedaje się pamięci o pojemności 16 Kb; rozpoczyna się wprowadzanie pamięci 64 Kb SRAM.**
- **Znacznie wzrasta udział mikroprocesorów 16-bitowych w produkcji mikroprocesorów wszystkich typów. Całkowita produkcja mikroprocesorów wyniosła w 1981 roku — 98 mln sztuk, a w 1982 — 164 mln. Są one masowo stosowane w sprzęcie powszechnego użytku i w samochodach.**
- **Rozpoczęła się masowa produkcja mikroprocesorów 8-bitowych wykonanych technologią CMOS; opracowano wersje CMOS mikroprocesorów typów: 8085, M6802, Z80. Zaczynają one przeważać wśród mikroprocesorów 8-bitowych.**
- **Znacznie wzrasta produkcja komputerów osobistych. W roku 1982 sprzedano 650 tys., w 1983 oczekiwano sprzedaży 1,1 mln egzemplarzy.**

Od dwóch lat toczy się na świecie dyskusja na temat zalet i wad przeciwstawnych kierunków w budowie mikroprocesorów, tzw. RISC (ang. reduced instruction set computer) i CISC (ang. complex instruction set computer).

Głównym celem architektury typu RISC jest dostarczenie małej liczby bardzo szybkich rozkazów. Przykładem realizacji tej zasady może służyć mikroprocesor opracowany na Uniwersytecie Kalifornijskim w Berkeley. Wykonuje on zaledwie 31 rozkazów, z tego 10 dotyczy pamięci. Długość słowa wynosi 32 bity. Wszystkie rozkazy odnoszące się do pamięci wykonywane są w dwóch cyklach maszynowych, pozostałe — w jednym cyklu. Liczba niezbędnych dostępów do pamięci jest zredukowana przez wprowadzenie zespołu 143 rejestrów, przechowujących dane, którymi się operuje. Czas wykonywania większości rozkazów, nie wymagających dostępu do pamięci, określony jest przez czas odczytu i sumowania zawartości dwóch rejestrów oraz zapisu wyniku do rejestru. Stosuje się zasadę pobierania następnego rozkazu podczas wykonywania bieżącej operacji (ang. pipelining).

Mikroprocesory zbudowane zgodnie z kierunkiem CISC wyposaża się w bardzo rozbudowaną listę rozkazów, zarówno co do liczby rozkazów, jak i ich złożoności. Operuje się danymi o różnego rodzaju strukturze i stosuje różne sposoby adresowania. Mikroprocesory takie budowane są z myślą o ułatwieniu implementacji języków wysokiego poziomu i systemów operacyjnych. Przedstawicielami tego kierunku są 32-bitowe mikroprocesory, które pojawiły się na rynku w 1982 roku — jednoukładowe HP 32 (HEWLETT-PACKARD) i BELLMAC 32A (BELL LABORATORIES) oraz rodziny NS16000 (NATIONAL SEMICONDUCTOR CORPORATION) i NCR/32 (NCR CORPORATION).

Dla porównania z mikroprocesorami dotychczas stosowanymi warto podać podstawowe parametry mikroprocesora 32-bitowego, jak np. HP 32:

- **liczba rozkazów — 230**
- **liczba rejestrów ogólnego zastosowania — 28**
- **czas dodawania dwóch liczb całkowitych 32-bitowych — 390 ns**
- **czas dostępu do pamięci dla słowa 32-bitowego — 560 ns**
- **liczba tranzystorów — 450 tys.**
- **liczba końcówek — 83**
- **częstotliwość zegara — 18 MHz.**

Kierunkiem rozwijanym niezwykle intensywnie przy użyciu techniki mikroprocesorowej jest grafika komputerowa. Pomimo szerokiej dostępności monitorów ekranowych o dobrej rozdzielczości, wyposażonych w kolor i grafikę liniową, nie wykorzystywano w pełni stwarzanych przez nie możliwości. Używa się ich w sposób tradycyjny, traktując je często jako przyspieszoną wersję dalekopisu. Tymczasem uzyskane — dzięki zastosowaniu mikroprocesorów i wykorzystaniu ich inteligencji — możliwości: selektywnej modyfikacji pewnych fragmentów obrazu, doboru wielkości znaków i ich koloru, stosowania ciemnych napisów na jasnym tle obok napisów normalnych, wykreślania schematów blokowych, a także duża szybkość pisania — pozwalają na znacznie lepszą komunikację systemu z użytkownikiem. W celu pełniejszego informowania użytkownika, który często nie jest zawodowym programistą, można:

— zastąpić znaki informacyjne i skróty, wymagające użycia podręcznika systemu do ich rozszyfrowania, przez napisy, które same się tłumaczą

— przy korzystaniu z oprogramowania systemowego w różnych trybach pracy, gdy trudno zapamiętać istniejące współzależności — podawać informacje o możliwościach, które istnieją w danej chwili oraz co należy zrobić, aby z nich skorzystać (podawanie tzw. menu)

— przy translacji, stosując podział ekranu na części — jednocześnie podawać tekst źródłowy, wynik translacji oraz informację o błędach, a kolorami wyróżniać słowa zarezerwowane, nazwy procedur, typy danych, poziomy zagnieżdżenia procedur (kolor powinien wybierać się automatycznie)

— korzystając z programu uruchomieniowego, po zatrzymaniu na pułapce — wyświetlać graficzny obraz rejestrów mikroprocesora z wpisaną zawartością i podawać informację, co trzeba zrobić, aby zmienić zawartość rejestru, która natychmiast po wykonaniu tej czynności jest na ekranie aktualizowana

— po zainicjowaniu zadania podawać graficzną informację o jego przebiegu.

*

Wszystkie wymienione kierunki są szczególnie intensywnie rozwijane w krajach Europy Zachodniej, w USA oraz Japonii, która stała się światową potęgą w dziedzinie mikroelektroniki (23% produkcji w latach 1982—1983 — w porównaniu do 20-procentowego udziału Europy). Produkcja charakteryzuje się tam szybkim wzrostem, wynoszącym 20% rocznie — pięciokrotnie szybszym niż dla przemysłu stalowego. Przewiduje się, że mikroelektronika stanie się w niedalekiej przyszłości wiodącym przemysłem japońskim, a Japonia — być może — przodującą potęgą w tej dziedzinie.

Osoby, które pragną uzyskać informacje bardziej szczegółowe, mogą je znaleźć w materiałach sympozjum, które zawierają pełne teksty wygłoszonych referatów (dwa tomy o łącznej objętości ok. 950 stron).

ANDRZEJ ŁAZARKIEWICZ
Instytut Problemów Jądrowych
Świerk

Ogłoszenia drobne (każde słowo — 20 zł, + 75 zł za egzemplarz pisma) publikujemy w pierwszej kolejności. Za ich treść i stylistykę odpowiadają zleceniodawcy, niemniej w przyszłości będziemy starali się publikować wyłącznie teksty w języku bardziej przypominającym polski. Autorów prosimy o kontakt z redakcją.

Cena większych ogłoszeń zależy od ich objętości: cała strona pisma — 22 tys. zł, 3/4 str. — 20 tys., 1/2 str. — 14 tys., 1/4 str. — 9 tys., 1/8 str. — 6 tys.

Od następnego numeru informacje natury handlowej publikujemy wyłącznie odpłatnie. GIEŁDĘ INFORMACJI rezerwujemy dla doniesień związanych z organizowaniem informatyki krajowej oraz dla ofert nie mających handlowego charakteru. (Red.)

GIEŁDA INFORMACJI

PRACA

Wyższa Szkoła Marynarki Wojennej im. Bohaterów Westerplatte w Gdyni zatrudni informatyków i automatyków do realizacji prac naukowo-badawczych (systemy czasu rzeczywistego). Dokładne informacje można uzyskać telefonicznie: 27-26-95.

KONFERENCJE

Systemy mikroprocesorowe

W dniach 17—19 listopada ub.r. w Płowdiw w Bułgarii odbyła się szósta z kolei międzynarodowa konferencja naukowo-techniczna z serii „Technika obliczeniowa'83" pod hasłem „Systemy mikroprocesorowe". Organizatorami konferencji były: sekcja informatyczna bułgarskiego stowarzyszenia inżynierów elektroniki, elektrotechniki i telekomunikacji oraz bułgarskie resorty i placówki naukowe związane z rozwojem informatyki. W imprezie wzięło udział ok. 180 specjalistów, w tym ok. 30 uczestników z zagranicy, głównie ze Związku Radzieckiego, NRD, Czechosłowacji, Węgier i Polski. Delegacja polska liczyła 9 osób.

W otwarciu oraz posiedzeniu plenarnym konferencji wzięli udział również uczestnicy odbywającego się równocześnie w Płowdiw posiedzenia Rady Głównych Konstruktorów Systemu Małych EMC, działającej w ramach Komisji Międzyrządowej ds. Współpracy Krajów Socjalistycznych w zakresie Elektronicznej Techniki Obliczeniowej (MK ETO). Przewodniczący tej Rady, Generalny Konstruktor SM EMC — prof. Naumow (ZSRR) podkreślił, że systemy mikroprocesorowe wchodzą w zakres prac Rady i są obecnie jednym z najważniejszych kierunków rozwoju SM EMC.

Na konferencję zgłoszono ok. 130 referatów, głównie z Bułgarii, które w wielu przypadkach stanowiły omówienie różnych fragmentarycznych zagadnień tego samego systemu mikroprocesorowego. Z powodu nieobecności wielu autorów wygłoszono jednak tylko ok. 60% referatów gospodarzy. Znaczną liczbę referatów wygłosili reprezentanci ZSRR i NRD, natomiast pojedyncze — Grecji, Węgier, Czechosłowacji i Polski. Program konferencji obejmował posiedzenie plenarne oraz obrady w pięciu sekcjach problemowych.

W ramach posiedzenia plenarnego wygłoszono następujące referaty:

- **„Problemy opracowania użytkowych systemów mikroprocesorowych" (J. Julzarij i inni — Bułgaria), zawierający przegląd 10 typów specjalizowanych mikrokomputerów, przygotowywanych do produkcji w Bułgarii, a przeznaczonych do zastosowań ekonomicznych, w handlu, do przetwarzania testów oraz do przygotowania danych.**

- **„Współczesna architektura systemów operacyjnych dla mikroprocesorów" (G. U. Fak — NRD), dotyczący zastosowania mikroprocesora 16-bitowego typu INTEL 8086,**

- **„Wykorzystanie mikroprocesorów przy projektowaniu sprzętu" (G. Kezling — ZSRR), omawiający realizowane w ZSRR zastosowanie mikroprocesorów do sterowania maszyn i urządzeń, robotów przemysłowych, w energetyce jądrowej, do automatyzacji pomiarów i testowania wyrobów, a także w biurach konstrukcyjnych i technologicznych pod postacią komputerów osobistych,**

- **„Przetwarzanie równoległe w systemach wieloprocesorowych" (J. Suchow — ZSRR), nawiązujący do współczesnych rozwiązań systemów komputerowych obsługujących wiele równoległych procesów obliczeniowych przez znaczną liczbę mikroprocesorów. Ma to szczególne znaczenie dla zwiększenia niezawodności sterowania procesami technologicznymi.**

Referaty wygłoszone w sekcjach dotyczyły następujących problemów systemów mikroprocesorowych:

- organizacji, struktury, elementów i metodyki projektowania (sekcja 1),
- algorytmów, obliczeń, modeli i techniki przetwarzania danych (sekcja 2),
- zastosowań (sekcja 3),
- oprogramowania (sekcja 4),
- testowania, diagnostyki i niezawodności (sekcja 5).

Powyższa problematyka dotyczyła mikroprocesorów 8-bitowych (INTEL 8080/85 , MOTOROLA 6800), 16-bitowych (INTEL 8086, ELEKTRONIKA 60), a nawet 32-bitowych (JAPX 432), jak również mikroprocesorów bipolarnych segmentowych (AM-2900, MOTOROLA 10800, INTEL 3000).

Treść szeregu referatów z ZSRR, NRD i Bułgarii, zwłaszcza dotyczących mikroprocesorów 16-bitowych oraz systemów wielomikroprocesorowych, wywoływała wrażenie większego w porównaniu do Polski zaawansowania tych krajów w opanowaniu techniki mikroprocesorowej, a zwłaszcza stworzenia tam mocnej podbudowy teoretycznej. Jak już wspomniano, wiele referatów bułgarskich dotyczyło różnych zagadnień (fragmentów) tego samego systemu mikroprocesorowego. Fragmenty te zostały rozpracowane przez różne, kilkuosobowe zespoły autorskie, co świadczy o istnieniu w tym kraju licznych wyspecjalizowanych grup naukowców, koncentrujących swe zainteresowania na celowo dobranych i dobrze skoordynowanych tematach prac badawczo-rozwojowych.

(TP)

KONFERENCJE

DPD'83

Szósta międzynarodowa konferencja nt. systemów transmisji danych — DPD'83 (Dálkový Přenos Dat) odbyła się w dniach 3—6 października 1983 r. w Karlovych Varach. Konferencje DPD są organizowane od 1970 r. przez czechosłowackie stowarzyszenie rozwoju techniki (ČSVTS) przy udziale zainteresowanych organów administracji państwowej i placówek naukowych oraz innych stowarzyszeń CSRS. Konferencje te stały się nie tylko międzynarodowym forum wymiany doświadczeń, lecz także stanowią doskonałą ilustrację aktualnego stanu badań i zastosowań technik sieciowych w różnych krajach.

W konferencji DPD'83 uczestniczyło ok. 400 osób, w tym 40 z zagranicy (10 z Polski, 9 z ZSRR, 8 z NRD, 7 z Bułgarii, 5 z Węgier oraz 1 z Austrii). Ogółem wygłoszono 68 referatów, z tego 32 przez uczestników zagranicznych. Dyskusje nad referatami poszczególnych bloków tematycznych prowadzone były w formie seminaryjnej, co znacznie ułatwiło konwersację z autorami oraz wymianę doświadczeń i poglądów, a także nawiązywanie współpracy między zespołami różnych ośrodków naukowych.

W odróżnieniu od poprzednich konferencji, w treści referatów dominowały zagadnienia o charakterze eksperymentalnym i praktycznym. Wskazywało to na postęp w porównaniu do treści poprzednich konferencji, na których dominowały prace o charakterze koncepcyjnym oraz badania modelowe oparte na metodach formalnych. Tym razem znacznie szerzej prezentowano problematykę lokalnych i publicznych sieci komputerowych oraz kierunki prac normalizacyjnych. W referatach gospodarzy przedstawiono doświadczenia uzyskane w budowie i próbnej eksploatacji pilotowej sieci komputerowej z węzłami komunikacyjnymi zlokalizowanymi w Bratysławie i Pradze. Większość referatów polskich dotyczyła doświadczeń uzyskanych w toku projektowania i budowy międzyuczelnianej sieci komputerowej MSK. W treści referatów gospodarzy i specjalistów z Polski znalazły swój wyraz wyniki wieloletniej współpracy między ośrodkami wiodącymi w budowie sieci komputerowych obu krajów — Politechniki Wrocławskiej oraz Instytutu Cybernetyki Stosowanej w Bratysławie.

Specjaliści z NRD przedstawili wyniki badań i zamierzenia rozwojowe sieci komputerowej DELTA. W referatach specjalistów z ZSRR, Węgier i Bułgarii dominowała tematyka transferu danych oraz metod optymalizacji wykorzystania zasobów w systemach wielodostępnych i środowiskowych sieciach komputerowych, opartych na komputerach i minikomputerach Jednolitego Systemu.

Przebieg konferencji ukazał liczne i wielokierunkowe osiągnięcia krajów z nami sąsiadujących. Szczególnie widoczna była szybko postępująca integracja technik przesyłania i przetwarzania danych oraz zdominowanie technik transmisji danych przez sieci komputerowe i systemy przetwarzania rozproszonego z bazami danych.

Problematyka konferencji koncentrowała się wokół zagadnień architektury oraz wydajności systemów transferu i przetwarzania danych. Wyniki prezentowanych badań dotyczyły wszystkich aspektów tej problematyki, od metod i środków wzrostu efektywności do metod i kryteriów oceny skutków ekonomicznych zastosowania sieci komputerowych. Zajmowano się więc pomiarami wydajności, projektowaniem sieci, technikami modelowania i symulacji oraz metodami predykcji skuteczności użytkowej projektowanych sieci.

Obrady prowadzono w sesjach plenarnych i problemowych. Tematyka sesji plenarnych dotyczyła podstaw teoretycznych oceny i optymalizacji efektywności transferu i przetwarzania danych oraz doświadczeń i wyników uzyskanych w projektowaniu i wdrażaniu sieci komputerowych w CSRS, PRL, WRL i ZSRR. Szczególną uwagę zwracano na zakres, formy i wyniki dotychczasowej współpracy zespołów specjalistycznych zajmujących się budową podstawowych komponentów sieci oraz uruchamianiem i eksploatacją sieci pilotowych w poszczególnych krajach. Omawiano także ważniejsze doświadczenia, uzyskane w toku realizacji eksperymentalnych usług sieciowych we współpracy z IIASA w Wiedniu.

Przedmiotem pięciu sesji problemowych były:

- teoretyczne zagadnienia miar i oceny efektywności,
- wydajność i skuteczność zastosowań technik transferu danych w sieciach komputerowych,
- efektywność sieci komputerowych w fazie projektowania i użytkowania,
- efektywność środków sprzętowych i programowych oraz metody optymalizacji,
- kierunki i możliwości nowych zastosowań w dziedzinie technik sieciowych.

Przegląd prezentowanych na konferencji osiągnięć posłużył do zidentyfikowania i określenia wielu dotąd nierozwiązanych jeszcze zadań. Dotyczą one przede wszystkim zagadnień organizacyjnych, prawnych i usługowych w dziedzinie wymiany i przetwarzania informacji. Istotnego znaczenia nabierają także nowe warunki występujące w stosunkach odbiorcy i dostawcy informacyjnych usług sieciowych. Najważniejsze zadania zostały sformułowane i skierowane w formie postulatywnej do odpowiednich krajowych i międzynarodowych organów państwowych i zawodowo-społecznych. Ze względu na fakt, że techniki transferu danych coraz bardziej determinują rozwój informacyjnych usług sieciowych uczestnicy konferencji sformułowali m.in. następujące wnioski:

- aktualność i perspektywiczność problematyki konferencji DPD, a zwłaszcza jej znaczenie dla kształcenia specjalistów w dziedzinie sieci komputerowych wskazują na potrzebę kontynuowania tego typu spotkań w okresach dwuletnich,
- pilnym zadaniem staje się ukierunkowanie współpracy organizacji zawodowych i szkół wyższych na opracowanie pojęć i ujednolicenie definicji, dotyczących technik pomiarowych oraz kryteriów i zakresu oceny efektywności systemów transferu i przetwarzania danych w fazach ich projektowania, budowy i użytkowania,
- konieczne jest przyspieszenie uruchomienia produkcji oraz zapewnienie w ramach kooperacji międzynarodowej dostaw urządzeń dostosowanych do większych szybkości transmisji danych, jak np. modemy typu MDS,
- celem zwiększenia dostępności usług sieciowych, zwłaszcza w zakresie gromadzenia i transferu danych, pilne jest opracowanie i uruchomienie produkcji typoszeregu terminali, których oprogramowanie umożliwiałoby ich instalowanie w różnego typu sieciach komputerowych. Produkcja takich terminali powinna być opłacalna w sensie ekonomicznym, a ich ceny — sprzyjać upowszechnieniu zastosowań sieci komputerowych,
- należy zwiększyć odpowiedzialność dostawców elementów oraz składowych systemu transferu i sieci komputerowych za zgodność jakościową produktów i efektywność usług z dokumentacją projektową i standardami tych systemów,
- należy wezwać wszystkie instytucje i organizacje zajmujące się budową sieci komputerowych do zjednoczenia wysiłków i środków w celu wypracowania zasad prawnych i regulaminów działalności służb łączności odpowiedzialnych za stronę ilościową i jakościową usług sieciowych,
- należy wzmocnić udział organów łączności w intensyfikowaniu prac zmierzających do wspólnego z użytkownikami opracowania koncepcji, projektów oraz zasad instalowania środków transferu danych, spełniających przyjęte standardy budowy i eksploatacji powszechnie dostępnych sieci komputerowych.

Organizatorzy jak zwykle opublikowali treść wygłoszonych referatów w języku angielskim i rosyjskim. W jednym z zeszytów materiałów konferencyjnych opublikowano również metodykę projektowania systemów transmisji danych.

Należy podkreślić znaczenie i rolę konferencji DPD jako jednej z bardzo nielicznych międzynarodowych imprez organizowanych w krajach socjalistycznych na temat sieci komputerowych. Ostatnia konferencja dowodzi o istotnych zmianach, polegających na wzroście doświadczeń, rozszerzeniu zakresu pracy badawczych i potencjału twórczego oraz rozwoju zastosowania systemów mikro- i minikomputerowych.

MIECZYSŁAW BAZEWICZ

Mikro-maraton

Angielski tygodnik popularno-naukowy NEW SCIENTIST relacjonował w kolejnych numerach przebieg „mikro-maratonu", który odbył się w Londynie w sierpniu ubiegłego roku. Maraton miał trwać tydzień i wykazać stopień niezawodności stających do współzawodnictwa 16-bitowych komputerów osobistych. Już po 24 godzinach jedna z maszyn została całkiem wyeliminowana z dalszych rozgrywek, a druga wymagała interwencji obsługi technicznej średnio raz na dwie godziny.

Sześć różnych typów komputerów podjęło wyzwanie rzucone przez firmę MICRO NETWORKS, która sprzedaje model o nazwie SAMURAI S-16. Firma twierdzi, że komputer ten może działać bez przerwy przez co najmniej dziewięć dni. Chociaż MICRO NETWORKS zaprosiła do udziału w maratonie około 30 firm, na starcie pojawiły się tylko IBM, OLIVETTI, WANG, COMART i LSI. Każdy z komputerów osobistych wykonywał program zawierający przeróżne procedury sortowania. Maszyny te miały przeredagowywać listy danych wejściowych, kopiować je z dysku na dysk, przeprowadzić kontrolę danych i skasować kopię.

Na realizację całego zadania komputery potrzebowały od sześciu minut do półtorej godziny. Jeden z zaproszonych arbitrów, redaktor Richard King z pisma PERSONAL COMPUTER NEWS, oświadczył, że jest zaskoczony różnicami w szybkości pracy poszczególnych maszyn. Ocenił, że najwolniejszy komputer zdoła w ciągu tygodnia wykonać program 200 razy, podczas gdy najszybszy — aż 1400. Po kilku dniach OLIVETTI M20, SAMURAI S-16, COMART COMMUNICATOR i IBM PERSONAL COMPUTER pozostały daleko w tyle za WANG i LSI M-FOUR.

Komputery firmy IBM miały poważne kłopoty. Jeden został zdyskwalifikowany, ponieważ odmówił wczytania programu. Organizatorzy podejrzewali, że było to spowodowane awarią przełącznika w mechaniźmie napędowym dysku. Drugi wymagał restartów co 2 godziny i 40 minut z powodu błędu w oprogramowaniu. Zgodnie z regulaminem, maszyna mogła być naprawiana w przypadku pojawienia się drobnych błędów, ale pod warunkiem, że przerwa w pracy nie przekroczy pół godziny, a zasilanie będzie wyłączone na co najwyżej pięć minut. Inżynierowie opiekujący się komputerami udzielali im takiej pomocy, o jakiej nie marzy normalny użytkownik. Tak traktowane systemy mają wszelkie szanse na wieloletnie sprawne funkcjonowanie.

Richard King stwierdził, że największym problemem dla „zawodników" było ciepło. Obwody przeciążone wykonywaniem przez komputer wciąż tych samych czynności mogą być przyczyną zakłóceń, chociaż cały sprzęt jest pozornie gotowy do pracy.

Producenci komputerów przeprowadzają próby poszczególnych elementów sprzętu, natomiast testy wytrzymałościowe sprzedawanych maszyn należą do rzadkości. Organizatorzy maratonu skrzętnie notowali informacje o wszystkim, co wydarzyło się w ciągu tygodnia w nadziei, że uzyskają wyraźniejszy obraz najsłabszych stron testowanych komputerów.

Maraton zakończył się nieco wcześniej niż przewidywano — organizatorzy musieli zostawić uczestnikom trochę czasu na spakowanie sprzętu. Do końca wytrwało tylko sześć maszyn z jedenastu rozpoczynających zawody. Zdaniem sędziów najlepsze były OLIVETTI M20 i SAMURAI S-16. Każdy z tych modeli był reprezentowany przez dwie maszyny i wszystkie cztery ukończyły maraton bez najmniejszych potknięć. Do mety dotarł również jeden komputer WANG i jeden LSI. Komputery IBM, LSI i COMART miały kłopoty na całej trasie, ale dzięki sprawności obsługi technicznej zdołały ukończyć zawody. Najwięcej pracy wykonała maszyna WANG, a jeden przebieg jej programu był pięć razy szybszy niż SAMURAI.

Po zakończeniu maratonu organizatorzy sprawdzili, czy programy i dane były identyczne dla wszystkich komputerów i czy uczestnicy nie stosowali zabronionych regulaminem „środków dopingujących". **K. I**

KALENDARZ

Lipiec

9—12, Las Vegas (USA): National Computer Conference (NCC) — krajowa konferencja i wystawa na temat mikroinformatyki, urządzeń peryferyjnych i oprogramowania — organizator: American Federation of Information Processing Societies

20—2.8, Montreal (Kanada): VII międzynarodowa konferencja na temat rozpoznawni obrazów — organizator: International Association for Pateern Recognition

23—27, Minneapolis (USA): XI konferencja i wystawa na temat grafiki komputerowej i technik interakcyjnych SIGGRAPH'84 — organizatorzy: ACM Special Interest Group on Computer Graphics oraz IEEE Technical Committee on Computer Graphics

Sierpień

20—28, Montreal (Kanada): VII międzynarodowa konferendowe sympozjum statystyki obliczeniowej — organizator: IASC

28—30, Kopenhaga (Dania): EUROMICRO'84 — 10. międzynarodowe sympozjum na temat mikroinformatyki i mikroprogramowania — organizator: EUROMICRO

Wrzesień

2—7, Weimar (NRD): IKM — X międzynarodowa konferencja na temat zastosowania matematyki w pracach inżynierskich — organizator: Hochschule für Architektur und Bauwesen Weimar

3—17, Londyn: INTERACT'84, międzynarodowa konferencja na temat czynnika ludzkiego w systemach informatycznych oraz współdziałania człowiek-maszyna — organizatorzy: IFIP, IFAC, IFORS oraz IEA

10—14, Paryż: VI międzynarodowy kongres cybernetyki i teorii systemów — organizator AFCET oraz World Organization of General Systems and Cybernetics

17—21, Paryż: międzynarodowa konferencja „Convention Informatique'84" oraz wystawa „SICOB'84"

18. Lipsk (NRD): międzynarodowe sympozjum na temat problemów szkolenia i dokształcania kadr dla mini- i mikrokomputerów — organizator: Schulungszentrum Robotron

18—21, Wrocław: krajowa konferencja „Mikrokomputery w automatyce i technice systemów" — organizator: Instytut Sterowania i Techniki Systemów Politechniki Wrocławskiej

24—28, Hong Kong: międzynarodowa konferencja pn. „Technologia informatyczna — środek maksymalizacji potencjału gospodarczego krajów azjatyckich" — organizator: SEARCC (South East Asia Regional Computer Confederation)

Październik

8—13, Warna (Bułgaria): VII międzynarodowe seminarium na temat systemów zarządzania bazą danych — organizator: INTERPROGRAMMA

25—30, Kolonia (RFN): V międzynarodowa wystawa „Orgatechnik Köln'84" — organizator: Messe- und Austellungs — Gesellschaft mbH Köln

Konieczność kompromisu

Treść rubryki TERMINOLOGIA w poprzednim numerze sprowokowała mnie do przedstawienia poglądu na temat kształtowania prawidłowych terminów informatycznych. Nie oznacza to jakiegokolwiek zamiaru rozpoczynania polemiki z aktualnym autorem tej rubryki red. Januszem Zalewskim, którego cenię za wytrwałe kontynuowanie tej niezwykle ważnej tematyki oraz znaczący wkład do uporządkowania, sprecyzowania i utrwalenia wielu istotnych pojęć. Do przekazania poniższych uwag czuję się upoważniony nie tylko ze względu na fakt, że w 1975 r. byłem inicjatorem i autorem tej rubryki na łamach naszego czasopisma, ale również z racji częstych kontaktów z autorami publikacji informatycznych, podczas których miałem okazję niejednokrotnie stwierdzić brak akceptacji niektórych propozycji red. Zalewskiego przez wielu przedstawicieli środowiska.

Uważni czytelnicy omawianej rubryki mogą stwierdzić coraz silniejsze dążenie red. Zalewskiego do tworzenia terminów autentycznie polskich z równoczesną systematyczną eliminacją wszelkiego rodzaju zapożyczeń z języka angielskiego. Poglądy swoje red. Zalewski wyraża jednak moim zdaniem w tonie zbyt autorytatywnym, w czym upodabnia się nieco do przedstawicieli krańcowo odmiennego stanowiska, tzn. osób przyjmujących jako zasadę bezpośrednie wprowadzanie do języka informatycznego terminów angielskich jako jedynie dobrze zrozumiałych przez każdego specjalistę. Z podejściem takim, podobnie jak red. Zalewski, oczywiście nie zgadzam się, mimo że praktykuje je niemała część środowiska naukowego. Jest to zjawisko występujące nie tylko w naszym kraju i wynika z szybkiego tempa przenoszenia osiągnięć amerykańskich. Tempo to powoduje przenikanie do piśmiennictwa specjalistycznego określeń używanych przez naukowców w wewnętrznych kontaktach roboczych. Postępowanie takie wynika po prostu z braku czasu na tworzenie sensownych odpowiedników terminów angielskich w języku ojczystym, co jednak nie upoważnia do przenoszenia roboczego slangu do treści publikacji.

Podobnie jak red. Zalewski staram się w miarę swoich możliwości nie dopuszczać do rozpowszechniania terminów slangowych, a w bardziej drastycznych przypadkach — zdecydowanie je zwalczać. Uważam jednak, że przesadny puryzm językowy jest równie niepożądany i — jak wykazuje praktyka — powoduje często zgodnie z naturą ludzką reakcje i dążenia przeciwstawne, zakłócając naturalny cykl tworzenia powszechnie akceptowanej przez środowisko terminologii specjalistycznej.

Dlatego proponuję rozwiązania kompromisowe, czego nauczyłem się przed laty śledząc przebieg szczególnie długotrwałego powstawania międzynarodowych norm terminologicznych w dziedzinie informatyki, jakie prowadzi Mię-

Dańda J., Poznański Z.: Programowanie w języku PL/M (1)

INFORMATYKA 1984, nr 7, s. 2

Pierwsza część charakterystyki uniwersalnego języka wysokiego poziomu PL/M, przeznaczonego dla mikrokomputerów. Na podstawie przykładów, omówiono podstawowe konstrukcje tego języka.

Dworzecki J.: Biblioteka programów dla systemu CP/M

INFORMATYKA 1984, nr 7, s. 5

Charakterystyka najważniejszych pozycji istniejącego oprogramowania mikrokomputerów korzystających z systemu operacyjnego CP/M. Omówiono języki programowania, oprogramowanie narzędziowe oraz pakiety i programy użytkowe.

Rybus R.: Oprogramowanie systemu mikroprocesorowego bez pamięci masowej

INFORMATYKA 1984, nr 7, s. 8

Charakterystyka oprogramowania systemu mikroprocesorowego bez dostępu do pamięci masowej, zrealizowanego dla mikroprocesora MOTOROLA 6800. Omówiono tworzenie programu źródłowego, proces translacji oraz uruchamianie programu.

Trojniar W.: FORTH — język i system programowania (2)

INFORMATYKA 1984, nr 7 s. 10

Druga część charakterystyki języka i systemu programowania FORTH, obejmująca opis dalszych elementów konstrukcji języka oraz elementów systemu programowania i sposobu przygotowania programów.

Wilczek T.: Roboty (2). Urządzenia kroczące

INFORMATYKA 1984, nr 7 s. 21

Druga część charakterystyki rozwoju robotów przemysłowych, zawierająca omówienie rozwiązań w kategorii robotów kroczących, a zwłaszcza nowych możliwości ich sterowania w wyniku zastosowania mikrokomputerów.

Даньда Й., Познаньский З.: Программирование на языке PL/M (I)

INFORMATYKA 1984, № 7, стр. 2

Первая часть характеристики универсального языка высокого уровня PL/M, предназначенного для микро-ЭВМ. На базе примеров обсуждаются основные конструкции этого языка.

Двожецкий Й.: Библиотека программ для системы CP/M

INFORMATYKA 1984, № 7, стр. 5

Характеристика наиболее существенных элементов существующего программного обеспечения микро-ЭВМ, использующих операционную систему CP/M. Обсуждаются языки программирования, вспомогательные программы, а также пакеты и программы использования.

Рыбус Р.: Программное обеспечение микропроцессорной системы без массовой памяти

INFORMATYKA 1984, № 7, стр. 8

Характеристика программного обеспечения микропроцессорной системы без доступа к массовой памяти, разработанного для микропроцессора MOTOROLA 6800. Обсуждаются составление исходной программы, процесс трансляции и откладка программы.

Тройняр В.: FORTH — язык и система программирования (2)

INFORMATYKA 1984, № 7, стр. 10

Вторая часть характеристики языка и системы программирования FORTH, содержащая описание дальнейших элементов конструкции языка, а также элементов системы программирования и способа подготовки программы.

Вильчек Т.: Роботы (2). Шагающие устройства

INFORMATYKA 1984, № 7, стр. 21

Вторая часть характеристики развития промышленных роботов, содержащая обсуждение решений в категории шагающих роботов, в особенности новых возможностей их управления в результате применения микро-ЭВМ.

dzynarodowa Organizacja Normalizacyjna (ISO). Długotrwałość procesu powstawania wspomnianych norm wynikała z trybu ich ustanawiania, który zakładał konieczność szczegółowego uzgadniania stanowisk wszystkich zainteresowanych krajów członkowskich. Ze zrozumiałych względów nie jest to sprawa prosta, gdyż bardzo często wymaga żmudnego przełamywania silnie zakorzenionych już w poszczególnych krajach pojęć. Trudności uzgodnienia stanowisk spowodowały, że w normach tych przy zdecydowanej większości terminów występują co najmniej dwie równoważne nazwy. Rozwiązanie takie zostało przyjęte również przy opracowaniu w 1970 r. pierwszej, lecz dotąd nie zaktualizowanej (mimo założonego trzyletniego terminu jej ważności), Polskiej Normy „Nazwy i pojęcia podstawowe" (PN-71/T-01016). W normie tej przy uchwalaniu w toku ostrych dyskusji terminu **złącze** (sądzę, że nie mniej zrozumiałego od proponowanego przez red. Zalewskiego **sprzęgu**), wprowadzono również synonim **interfejs**. Jest to zgodne z wytycznymi ISO, która zaleca przyjmowanie do terminologii narodowych tych nazw, które w identycznym lub podobnym brzmieniu utrwaliły się już wcześniej w językach krajów przodujących w rozwoju technologii. Uzasadnieniem takiego podejścia są korzyści, jakie dzięki zbieżności terminów powstają w kontaktach i współpracy międzynarodowej specjalistów.

Moje osobiste doświadczenia wskazują również, że działanie wbrew istniejącym tendencjom i poglądom środowiska prowadzą do przykrych porażek. Klasycznym przykładem takiego przypadku był wprowadzony do wspomnianej Polskiej Normy **zapis** jako odpowiednik angielskiego terminu **record**. Podstawowymi argumentami wysuniętymi wówczas przeciw wprowadzeniu polskiej nazwy **rekord** było nie tylko szczególnie narzucające się skojarzenie z terminologią sportową, ale również brak zapożyczenia terminu angielskiego zarówno w języku francuskim (**enregistrement**), jak i niemieckim (**Satz**). Życie wykazało jednak, że zapis, mimo prowadzonej przez INFORMATYKĘ uporczywej walki o jego akceptację i utrwalenie, nie został przyjęty przez przytłaczającą część środowiska, która zaakceptowała szeroko już rozpowszechniony i ogólnie zrozumiały **rekord**.

Przytoczone przykłady wskazują, że w przypadku braku ostatecznej akceptacji przez środowisko należy dopuszczać co najmniej dwa równoważne terminy. Jedynie czas może wykazać, który termin uzyska społeczną aprobatę i stopniowo wyeliminuje konkurenta drogą naturalnej selekcji. Tak więc np. obok **zgodności** proponuję utrzymać nadal **kompatybilność**. Na poparcie takiego stanowiska chciałbym przypomnieć, że nawet Francuzi, którzy ze względu na swą anglofobię wzmogli w ostatnich latach walkę o wyeliminowanie z języka informatyków wszelkich amerykanizmów, akceptują m.in. **interface**. Przy tej okazji warto również zauważyć, że red. Zalewski proponując usunięcie słowa **kompatybilność**, lansuje równocześnie bez zastrzeżeń **implementację**. Apeluję więc, aby również w dziedzinie słownictwa informatycznego zrezygnować z przesadnego eksponowania czystości językowej, satysfakcjonującej głównie filologów.

WŁADYSŁAW KLEPACZ

Dańda J., Poznański Z.: Programming in the PL/M language (1)

INFORMATYKA 1984, No. 7, p. 2

First part of characteristics of the PL/M universal high level programming language for microcomputers. Illustrative examples for main components of the language's structure are discussed.

Dańda J., Poznański Z.: Programmierung in der PL/M Sprache (1)

INFORMATYKA 1984, Nr. 7, S. 2

Erster Teil einer Charakteristik von der universellen höheren Programmiersprache PL/M die für Mikrorechner bestimmt ist. Es wurden Grundkonstruktionen dieser Sprache charakterisiert.

Dworzecki J.: Programs library for the CP/M system

INFORMATYKA 1984, No. 7, p. 5

Characteristics of the main items from existing software for microcomputers with the CP/M operating system facility. Programming languages and tools, as well application packages and programs are discussed.

Dworzecki J.: Programmbibliothek für das CP/M System

INFORMATYKA 1984, Nr. 7, S. 5

Eine Charakteristik von wichtigsten Posten der bestehenden Mikrorechnersoftware, die mit Anwendung des CP/M Betriebssystems verbunden ist. Es wurden Programmiersprachen, Programmierungshilfsmittel sowie Anwendungspakete und -programme besprochen.

Rybus R.: Software for microprocessor system without mass store

INFORMATYKA 1984, No. 7, p. 8

Characteristics of software for microprocessor system without mass store, which was realized for MOTOROLA 6800 microprocessor. Source program building, translation process and program debugging are discussed.

Rybus R.: Software für ein Mikroprozessorsystem ohne Massenspeicher

INFORMATYKA 1984, Nr. 7, S. 8

Eine Lösung von Software für ein Mikroprozessorsystem ohne Massenspeicher, die für MOTOROLA 6800 Mikroprozessor realisiert wurde. Es wurden die Bildung des Quellprogramms, Übersetzungsprozess und Programmsprüfung besprochen.

Trojniar W.: FORTH — the language and programming system (2)

INFORMATYKA 1984, No. 7, p. 10

Second part of the FORTH language and programming system characteristics, which includes description of further language construction elements, as well as of programming system elements and program preparing method.

Trojniar W.: FORTH — Programmiersprache und -system (2)

INFORMATYKA 1984, Nr. 7, S. 10

Zweiter Teil einer Charakteristik von FORTH-Programmiersprache und -System, der eine Beschreibung weiterer Sprachekonstruktionselemente, sowie der Programmiersystemelemente und der Methode für Programmvorbereitung umfasst.

Wilczek T.: Robots (2). Walking machines

INFORMATYKA 1984, No. 7 p. 21

Second part of characteristics of industrial robots development, which includes presentation of walking robots solutions and their new control facilities achieved through microcomputer application.

Wilczek T.: Roboter (2). Schreitende Maschinen

INFORMATYKA 1984, Nr. 7, S. 21

Zweiter Teil einer Charakteristik von Industrieroboter entwicklung, die eine Besprechung von Lösungen der Schreitroboter, insbesonders ihrer neuen Steuermöglichkeiten mittels Mikrorechner, umfasst.

O najczęstszych błędach w terminologii mikrokomputerowej (2)

W bieżącym numerze kontynuuję omawianie błędów spotykanych w nazewnictwie stosowanym w technice mikroprocesorowej.

Pewien rodzaj często popełnianych błędów polega na nadawaniu pojęciu nazwy o nieodpowiednim znaczeniu. Może to wynikać z pomieszania znaczeń wyrazów bliskoznacznych, z niewłaściwego tłumaczenia terminów angielskich lub z innych przyczyn. Tak jest w przypadku **rozkazów** (ang. instruction) i instrukcji (ang. statement), warto więc zwrócić uwagę na to rozróżnienie. Nazwa **rozkaz** odnosi się tylko do komputera (procesora), tzn. do sprzętu, natomiast **instrukcja** — w zasadzie tylko do języka programowania, o czym nie wszyscy pamiętają.

Podobna jest przyczyna niezgodności w używaniu terminów **szyna** i **magistrala** (por. INFORMATYKA, nr 9—10, 1981). Wydaje się, że rozsądna jest następująca zasada:

- na oznaczenie połączenia (jak np. ścieżka, przewód), którym przesyłany jest pojedynczy sygnał — używać wyrazu **linia** (ang. line)
- na oznaczenie grupy linii spełniających pojedynczą funkcję — używać wyrazu **szyna** (np. szyna danych, szyna adresowa; ang. data bus, address bus)
- na oznaczenie wielofunkcyjnego zespołu szyn zapewniających pełną komunikację między jednostkami funkcjonalnymi, dołączonymi z zasady równolegle — używać wyrazu **magistrala** (ang. bus, highway, dataway itp.).

Tym śladem można podążać dalej i konsekwentnie definiować szersze pojęcia, określając — na przykład — istotną różnicę między **sprzęgiem** a **łączem**:

- **sprzęg** (ang. interface) jest zbiorem zasad określających sposób połączenia (a więc — magistralę), parametry wymienianych sygnałów i protokoły transmisji, umożliwiające współpracę różnych jednostek lub urządzeń
- **łącze** (ang. data link) jest to zestaw części dwóch urządzeń końcowych (ang. data terminal equipment, DTE), które są sterowane specjalnym protokołem i wspólnie z torem (ang. data circuit) umożliwiają przekazywanie danych.

Operując językiem teleinformatyki (a właściwie — telekomunikacji), można powiedzieć, że łącze składa się z toru transmisyjnego, tj. urządzeń komunikacyjnych (ang. data communications equipment, DCE), np. modemów, połączonych linią transmisji, oraz — z części urządzenia końcowego realizującej protokół. Obydwa urządzenia są połączone **stykiem** (ang. data transmission interface), gdyż tak w telekomunikacji nazywa się sprzęg. Tak więc określenie związków między **sprzęgiem, łączem** i **stykiem,** choć skomplikowane, nie jest niemożliwe.

Konieczność zachowania zgodności semantycznej nazwy z treścią pojęcia przesądza o tym, że nie należy mówić **dno stosu** lecz — **spód stosu** (ang. bottom of the stack), podobnie jak nie — **głębokość** lecz **wysokość stosu** (ang. depth of the stack). Przy tej okazji warto stwierdzić, że żądanie zgodności semantycznej nie jest kryterium wystarczającym do stwierdzenia poprawności pojęć: **wierzchołek stosu** oraz **szczyt stosu.** Jeżeli zastosujemy pomocniczą regułę — zakładającą, że nowe pojęcia nie powinny mieć nazw używanych w innym znaczeniu w dziedzinach pokrewnych — to nazwa wierzchołek wydaje się mniej odpowiednia, gdyż jest używana w geometrii (wierzchołek wielokąta) i w teorii grafów (wierzchołek grafu). Dlatego proponuję używać nazwy **szczyt stosu** (ang. top of the stack), mimo że nazwa **wierzchołek** stosu jest również zrozumiała.

Z językowego punktu widzenia niewiele sensu ma też sformułowanie **uzupełnienie do dwu** (ang. two's complement), gdy mowa o przedstawianiu liczb w kodzie uzupełnieniowym. Ciąg kodowy liczby zapisanej w tym kodzie jest uzupełnieniem dwójkowego ciągu odpowiadającego tej liczbie z przeciwnym znakiem, a nie uzupełnieniem samej liczby do 2, gdyż to nic nie znaczy. Wynika stąd jasno, że należałoby mówić **uzupełnienie dwójkowe,** a nie — **uzupełnienie do dwu.**

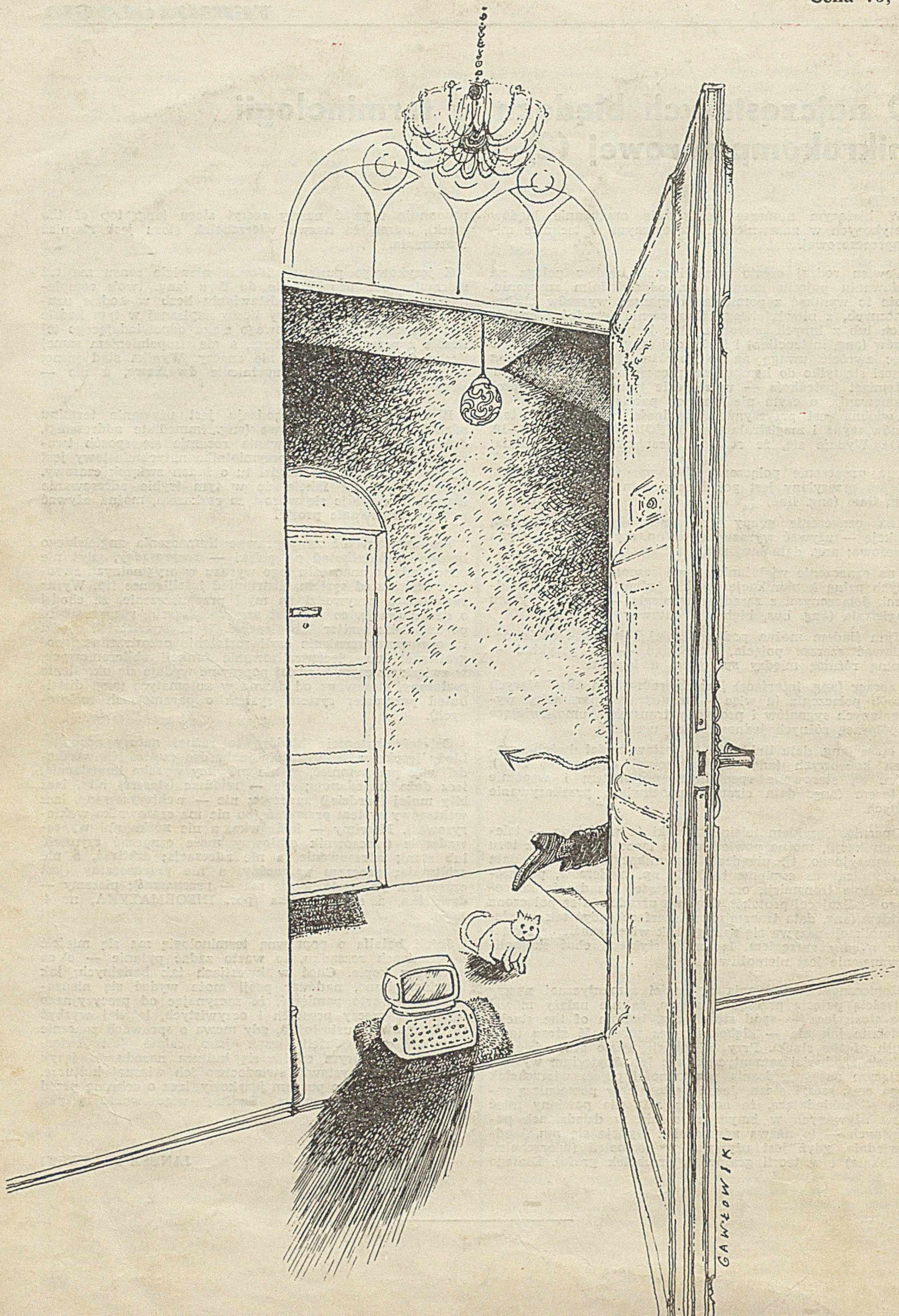
Błędem tego samego rodzaju jest używanie terminu **adresowanie natychmiastowe** (ang. immediate addressing). Jeżeli przez tryb adresowania rozumie się sposób tworzenia adresu, to użycie przymiotnika natychmiastowy jest niewłaściwe, gdyż nie chodzi tu o żaden związek czasowy. Opierając się na fakcie, że w tym trybie adresowania argument znajduje się wprost za rozkazem, można używać nazwy **adresowanie proste.**

Nie uważam też za poprawne tłumaczenia angielskiego imiesłowu **distributed** na polski — **rozproszony,** gdyż nie odpowiada to znaczeniu tego wyrazu w oryginalnych zwrotach **distributed system, distributed intelligence** itp. Wyraz **rozproszony** przywodzi na myśl przypuszczenie, że chodzi o dezintegrację, co nie jest prawdą. Zresztą, **rozpraszać** to po angielsku **scatter** lub **disperse.** W rzeczywistości, chodzi tu o rozprzestrzenienie czyli rozłożenie przestrzenne (geograficzne) lub zdecentralizowanie funkcji obliczeniowych, w związku z czym bardziej poprawne wydaje się określenie **rozłożony,** stosowane od dawna w automatyce (ang. distributed parameter system, system o parametrach rozłożonych).

Ostatnia kategoria błędów, to różnej natury pospolite błędy językowe. Spotyka się je bardzo często. Nie zaszkodzi więc przypomnieć, że nie mówimy **wysoka impedancja,** lecz **duża impedancja;** nie — **młodsze (starsze) bity,** lecz **bity mniej (bardziej) znaczące;** nie — **wektoryzowany,** lecz **wektorowy system przerwań** (bo nie ma czasownika wektoryzować). Mówimy — **końcówka,** a nie **nóżka** ani **wyprowadzenie** (rzeczownik odsłowny może oznaczać czynność lub stan); **adresowanie,** a nie **adresacja; arbitraż,** a nie **arbitracja; program przenośny,** a nie **przenaszalny** (jest czasownik **przenosić,** a nie ma — **przenaszać**); piszemy — **dyskietka,** a nie **dysketka** (por. INFORMATYKA, nr 4, 1983).

Jeżeli batalia o poprawną terminologię ma się mieścić w granicach rozsądku, to warto zadać pytanie — o co kruszymy kopie. Choć w kwestiach tak banalnych, jak terminologiczne, nadmiar pasji może wydać się nieuzasadniony, warto pamiętać, że zaczynając od precyzyjnego wyrażania rzeczy prostych i oczywistych, łatwiej wyzbyć się frazesów i pustosłowia, gdy mowa o sprawach znacznie poważniejszych. Jeżeli język jako całość jest zwierciadłem, w którym odbija się kultura narodu, to język informatyków wystawia świadectwo ich własnej kulturze. Nie chodzi mi tu o puryzm językowy, lecz o ochronę przed informatyczną nowomową. Bądźmy więc wobec języka bardziej pokorni.

JANUSZ ZALEWSKI


GAWŁOWSKI