4 1984

informatyka

LILITH i MODULA-2

Mikrokomputery – techniki interpretacji

BRNO'83

WYDAWNICTWO SIGMA

informatyka

4 1984

Nr 4
Miesięcznik Rok XIX
Kwiecień 1984

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

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-041 Warszawa, ul. Jasna 11/16, pok. 243 i 244, tel. 27-71-40 lub 26-82-61 w. 184

Zakł. Graf. „Tamka". Zam. 2054. Obj. 4,0 ark. druk. Nakład 4000 egz. T-46.

INDEKS 36124

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

WYDAWNICTWO SIGMA
NACZELNA ORGANIZACJA TECHNICZNA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Roman Żelazny o narzędziach inżynierii oprogramowania
- Cezary Zieliński i Teresa Wilczek o robotyce
- Tadeusz Szuba o związku PROLOGU z PASCALEM
- Wojciech Trojniar o języku i systemie programowania FORTH
- Marek Rakowski i Andrzej T. Rosiński o jednostrukturowych mikrokomputerach 8-bitowych
- Roman Trechciński o systemie MULTIBUS-II
- Krzysztof Rzymkowski o systemie modularnym VME
- Marek Pawłowski i Andrzej Woźniak o programatorze PROG-2
- Tadeusz Mazurkiewicz o regresie polskiej informatyki

WITOLD ABRAMOWICZ
Institut für Informatik
Eidgenossische Technische Hochschule
Zürich

MODULA-2 i LILITH
zgodność metod i narzędzi informatycznych

Gwałtowny rozwój technologii elektronicznej umożliwił budowanie małych komputerów (mikro-, mini-), małym zespołom konstruktorów, a nawet zaawansowanym hobbystom. Jednak budowa komputera nie powinna być końcowym celem, jaki przyświeca takim przedsięwzięciom. Komputer jest narzędziem, które w połączeniu z metodami informatycznymi umożliwia manipulowanie danymi nazywanymi raz komputerowym systemem wspomagania projektowania (CAD), innym razem systemem wyszukiwania informacji (IR), a jeszcze innym razem systemem finansowym. Komputer jest narzędziem, a w budowie systemów informatycznych wspierają go m e t o d y. Na poziomie połączeń z systemem liczącym uosobione są one głównie przez język programowania, kompilator i system operacyjny.

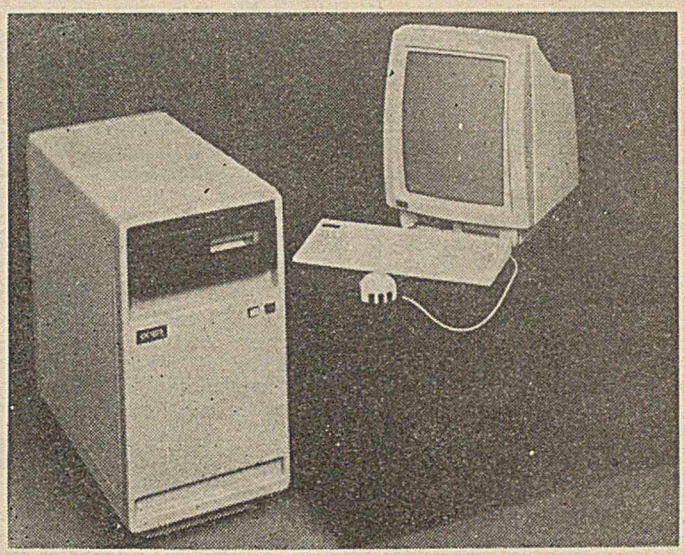
Łatwość i relatywnie mały koszt budowy lub kupna sprzętu oraz względnie duże koszty metod informatycznych powodują przypadkowość powiązań różnorodnych metod z narzędziami. Nie można myśleć o tworzeniu efektywnych systemów informatycznych, jeżeli metody i narzędzia są skojarzone na drodze przypadku. Pedantyczność w tym zakresie nie jest luksusem, natomiast bez wątpienia luksusem jest ignorowanie tej prawdy. Choć początkowe koszty systemu są niższe, jednak jego eksploatacja i ewentualne zmiany — dużo droższe. Jest to prawda mająca szczególne konsekwencje dla użytkownika systemu, mniejsze zaś dla jego producenta. Wskazanie zjawisk związanych z tym problemem spada zatem na ośrodki badawcze nie związane w sposób bezpośredni z wielkim przemysłem.

Do przeprowadzenia tego projektu niezbędny jest zespół naukowy silny merytorycznie, o możliwie szerokich zainteresowaniach naukowych, dający się jednak centralnie sterować w celu realizacji szeroko zakrojonego programu naukowego. Warunkom takim odpowiada Instytut Informatyki Politechniki Federalnej w Zurichu. Pracuje w nim ok. 50 pracowników naukowych zorganizowanych w grupy zajmujące się bardzo różnorodną tematyką. Instytut posiada jednak mechanizmy sprzyjające centralizacji niektórych poczynań.

Jesienią 1977 prof. Niklaus Wirth postanowił skonstruować komputer osobisty w powiązaniu z opracowaniem środków programowych i postawił przed swoim zespołem, liczącym w ostatnich latach przeciętnie ośmiu współpracowników, następujące cele:

- Opracowanie języka programowania ukierunkowanego na zastosowania w zakresie inżynierii oprogramowania. Planowanym sprzętem miał być komputer osobisty. Język ten otrzymał później nazwę MODULA-2 (MODUlar programming LAnguage).
- Skonstruowanie **kompilatora** tego języka.
- Zaproponowanie prostego **systemu operacyjnego.** Nie przewidywano tutaj jednoczesnego podziału zasobów między różnych użytkowników.
- Zaprojektowanie nowoczesnego i elastycznego **edytora tekstu.**
- Implementacje szeregu **programów pomocniczych** niezbędnych dla organizacji plików (są to programy służące do tworzenia, kopiowania i niszczenia plików).
- Programowanie zbioru **modułów bibliotecznych** umożliwiających dostęp do plików, współpracę z urządzeniami zewnętrznymi, organizację pamięci.
- Opracowanie **architektury systemu** jako optymalnego sprzęgu (ang. interface) pomiędzy sprzętem a kompilatorem. Oprogramowanie **interpretera mikrokodu.**
- Skonstruowanie maszyny i wyprodukowanie dwóch pierwszych egzemplarzy.
- Sprawdzenie prototypów, wprowadzenie zmian i wyprodukowanie serii na potrzeby Instytutu Informatyki ETH w Zurichu.

W końcu 1983 roku zadania te zostały w pełni zrealizowane. W Instytucie Informatyki ETH każdy pracownik, włącznie z pracownikami administracji oraz sekretariatu, dysponuje własnym komputerem osobistym LILITH — taką nazwę otrzymała nowa maszyna (fot. 1). Kilkanaście komputerów stoi również do dyspozycji studentów wyższych lat studiów informatycznych, którzy realizują na nich obowiązkowe prace semestralne i dyplomowe. Kształcenie propedeutyczne dla studentów wszystkich wydziałów — ponad 1000 osób — odbywa się na maszynach APPLE II w języku PASCAL.



Fot. 1. Komputer osobisty LILITH

O sukcesie przedsięwzięcia świadczy podjęcie produkcji komputera w celach handlowych przez firmę DISER AG w Szwajcarii. Maszyna znalazła już zastosowanie w dziedzinie komputerowego wspomagania projektowania i zarządzania.

LILITH umożliwił realizację wielu projektów w II ETH. Należy tutaj wymienić implementacje relacyjnego modelu baz danych LIDAS (zaprogramowanego w rozszerzonej wersji języka MODULA-2 — MODULA/R). Kompleksowym problemem rozwiązywanym za pomocą LILITH są także zagadnienia związane z systemami informacyjnymi. Opracowano systemy wyszukiwania informacji CALIBAN oraz system rozpowszechniania informacji ATHENE. Przeprowadzono także eksperymenty związane z dialogiem człowiek-maszyna, tworząc np. eksperymentalny system dialogowy XS-2. O szerokim zakresie tych prac świadczy projekt związany z muzyką komputerową.

Pierwszym zadaniem mającym prowadzić do celu wytyczonego przez prof. Wirtha, było opracowanie języka programowania. Jest on z założenia jedynym językiem dostępnym dla tej maszyny. Nie przewidywano opracowania asemblera. Nowy język musiał zatem odpowiadać wymaganiom, jakie stawia się przed językami wysokiego poziomu niezależnymi od maszyny, a także stawianym przed językami niskiego poziomu — np. dla procedur zarządzania pamięcią. Przy opracowywaniu nowego języka skorzystano z wielu doświadczeń związanych z PASCALEM, którego autorem jest również prof. Wirth. Język MODULA-2 stanowi rozszerzenie PASCALA o następujące elementy:

— koncepcję modułu jako podstawowej struktury oprogramowania; idea modułów definicyjnych i implementacyjnych rozwiązuje w dużej mierze konflikt pomiędzy niezależnością poszczególnych składników oprogramowania a ich integralnością; problematyka ta jest szczególnie istotna dla dużych projektów
— bardziej systematyczny zapis syntaktyczny, likwidujący pewne „przegadania" występujące w innych językach wysokiego poziomu
— kocepcje procesu jako punktu wyjścia do wielozadaniowości
— możliwość programowania w mikrokodzie, wynikająca z istnienia tylko jednego języka programowania
— wprowadzenie typu procedurowego umożliwiającego dynamiczne przyporządkowanie procedury zmiennej; umiejętne wykorzystanie tego mechanizmu może prowadzić do dużego wyrafinowania programów.

Architektura nowej maszyny miała wynikać z języka, dlatego też początkowe eksperymenty prowadzono na komputerze PDP 11/40 z pamięcią 28 K słów. Efektem tych działań jest handlowa dostępność kompilatora MODULI-2 dla szeregu różnych systemów (PDP 11, VAX — VMS i UNIX,APPLE/UCSD pSystem, INTEL 8086 i MOTOROLA 68000). Licencje na te kompilatory zakupiło kilkaset ośrodków akademickich i przemysłowych na całym świecie.

Koncepcja modułu jest kluczową dla języka MODULA-2. Moduł jest podstawową strukturą oprogramowania. Moduły mogą eksportować obiekty, takie jak procedury lub typy. Eksportowane obiekty są zgromadzone w definicyjnej części modułu — w module definicyjnym (ang. definition module). Inne moduły mogą importować eksportowane obiekty. Moduł definicyjny reprezentuje moduł we współpracy z innymi modułami. Moduł implementacyjny (ang. implementation module) zawiera natomiast metody prowadzące do otrzymania obiektów eksportowanych przez moduł definicyjny (nie wszystkie obiekty modułu muszą być eksportowane). W normalnej sytuacji moduły implementacyjne są nawidoczne dla programisty importującego obiekty z modułu definicyjnego. Opisaną metodę zilustrowano przykładem na rysunku 1.

Dalsze szczegóły o języku MODULA-2 można znaleźć w książce N. Wirtha [23], która zawiera oficjalny raport języka, a także szereg wskazań na temat systematycznego programowania. INFORMATYKA zapoznała już Czytelników z tym językiem w trzech poprzednich numerach.

Kompilator MODULI-2 oparto na zasadzie rozdzielonej kompilacji. Każdy moduł jest kompilowany osobno. Jej wynikiem jest m.in. **symbol-file** zawierający pełny opis obiektów zdefiniowanych w module definicyjnym. **Symbol-file** traktowany jest przez kompilator jako jedyny i jednoznaczny reprezentant modułu z punktu widzenia innych modułów importujących zdefiniowane w nim obiekty. Moduł definicyjny i implementacyjny są kompilowane niezależnie od siebie. Takie podejście umożliwia zmianę metod prowadzących do uzyskania eksportowanego obiektu bez kompilacji modułu definicyjnego, tak długo, jak długo zmiana metod nie powoduje zmiany definicji samego obiektu. Wynika stąd, że nawet zmiany metod uzyskania obiektów nie muszą prowadzić do zmian w programach wykorzystujących te obiekty, nie powodują także konieczności ich kompilacji. Jest to istotne dla modułów, które znalazły szerokie zastosowanie, takich, które są importowane przez liczne inne moduły.

Kompilacja odbywa się w czterech etapach, w pierwszym dokonuje się analizy syntaktycznej, w drugim analizuje się deklaracje, w trzecim sprawdza się zgodność typów danych w wyrażeniach, a w czwartym produkuje tzw. **M-kod.** W trakcie kompilacji z pliku zawierającego tekst modułu powstają: **listing file, symbol file, reference file** i **object file.**

```
DEFINITION MODULE PierwszyModul;

  EXPORT QUALIFIED
    PierwszaProceduraPierwszegoModulu,
    DrugaProceduraPierwszegoModulu,
    TypParametru;

  TYPE
    TypParametru = ..............;

PROCEDURE PierwszaProceduraPierwszegoModulu (Parametr :
TypParametru);

PROCEDURE DrugaProceduraPierwszegoModulu (VAR Parametr :
TypParametru);

END PierwszyModul.


IMPLEMENTATION MODULE PierwszyModul;

PROCEDURE PierwszaProceduraPierwszegoModulu (Parametr :
TypParametru);
BEGIN
      ..............
      ..............
      ..............
END PierwszaProceduraPierwszegoModulu;

PROCEDURE DrugaProceduraPierwszegoModulu (VAR Parametr :
TypParametru);
BEGIN
      ..............
      ..............
      ..............
END DrugaProceduraPierwszegoModulu);

..............
..............

END PierwszyModul.


DEFINITION MODULE DrugiModul;

  FROM PierwszyModul IMPORT
    TypParametru;

  EXPORT QUALIFIED
    PierwszaProceduraDrugiegoModulu,
    DrugaProceduraDrugiegoModulu;

PROCEDURE PierwszaProceduraDrugiegoModulu (Parametr : TypParametru);

PROCEDURE DrugaProceduraDrugiegoModulu (VAR Parametr : TypParametru);

END DrugiModul.


IMPLEMENTATION MODULE DrugiModul;

  FROM PierwszyModul IMPORT
    PierwszaProceduraPierwszegoModulu,
    DrugaProceduraPierwszegoModulu;

PROCEDURE PierwszaProceduraDrugiegoModulu (Parametr : TypParametru);
BEGIN
      ..............
      PierwszaProceduraPierwszegoModulu (Parametr);
      ..............
      ..............
END PierwszaProceduraDrugiegoModulu;

PROCEDURE DrugaProceduraDrugiegoModulu (VAR Parametr : TypParametru);
BEGIN
      ..............
      ..............
      DrugaProceduraPierwszegoModulu (Parametr);
      PierwszaProceduraPierwszegoModulu (Parametr);
      ..............
END DrugaProceduraDrugiegoModulu);
```

Rys. 1. Przykład ilustrujący koncepcję modułu

OPROGRAMOWANIE PODSTAWOWE

System operacyjny MEDOS-2 jest systemem otwartym, umożliwiającym użytkownikowi dostęp do wszystkich zasobów. Składa się on z trzech części: programu ładującego, systemu zarządzającego zbiorami oraz procedur współpracy z urządzeniami wejścia-wyjścia.

Głównym programem systemu operacyjnego jest interpreter poleceń. Czyta on i interpretuje połączenia (czyta nazwę programu i uaktywnia odpowiedni mu program), a także informuje o błędach w wykonywaniu bieżącego programu.

W przypadku akceptacji nazwy programu przez interpreter, program ładujący kopiuje odpowiedni moduł do pamięci operacyjnej (dokładnie tylko **object file**). Jeżeli dany moduł importuje obiekty z innych modułów, co jest regułą, dokonuje się również kopiowania importowanych modułów. Obowiązują tu podobne mechanizmy jak przy odwoływaniu się procedur do innych procedur, np. w PASCALU.

Bardzo duże znaczenie dla programisty ma system zarządzania plikami **FileSystem**. Wszystkie pliki dostępne w systemie są traktowane jako ciągi bajtów zapamiętane na pewnym nośniku informacji (dla systemu operacyjnego, na tym poziomie abstrakcji, nie jest istotne jakie jest to medium — musi on znać tylko jego nazwę). Moduł **FileSystem** jest sprzęgiem pomiędzy programistą a plikami, którymi chce on manipulować. Moduł jest odpowiedzialny za tworzenie, nazywanie, zapisywanie, odczytywanie, pozycjowanie oraz niszczenie plików. Każdy plik jest reprezentowany przez swoją nazwę. Dalsze informacje o plikach w trakcie manipulowania nimi zawiera związany z nimi opis (długość zbioru, obecna pozycja w zbiorze, ostatnio wykonywana operacja, nośnik na jakim się znajduje, itd.). Za funkcje związane z dostępem do plików na dysku i za organizację pamięci odpowiedzialny jest moduł **DiskSystem.**

Pakiet procedur współpracy z urządzeniami wejścia-wyjścia umożliwia dokonywanie bardzo różnorodnych operacji. Najprostsze są związane z wprowadzaniem informacji z klawiatury alfanumerycznej. Inne procedury są związane z urządzeniem wejściowym zwanym myszką (ang. mouse). Myszka jest urządzeniem umożliwiającym operatorowi wskazanie określonego punktu lub obszaru na monitorze (fot. 1 — poniżej klawiatury). Operator może dowolnie wybierać położenie punktu w obszarze, zdefiniowanym na monitorze ekranowym przez analityka i programistę systemu aplikacyjnego, przez zmianę położenia myszki na stole. O tym, jakiemu położeniu myszki odpowiada dany punkt monitora, informuje operatora kursor. Myszka jest zaopatrzona w czujnik przenoszący zmiany położenia myszki na stole na zmiany położenia wskaźnika na monitorze. Dokładność wskazań za pomocą myszki odpowiada praktycznie rozdzielczości monitora ekranowego i — w przeciwieństwie do pióra świetlnego — nie zależy od amplitudy, z jaką trzęsie się ręka operatora.

Zrealizowano także procedury umożliwiające wyprowadzanie informacji na monitor ekranowy. Najprostsze z nich pozwalają na przedstawianie znaków i tekstów. Maszyna LILITH jest wyposażona w graficzny monitor ekranowy oparty na technice **raster-scan**, a umożliwiający przedstawienie — w zależności od rodzaju (monitor horyzontalny lub wertykalny) — ok. 800 × 600 punktów. Umożliwia to programowe definiowanie różnych rodzajów znaków (rys. 2).

pokazowy tekst

pokazowy tekst

pokazowy tekst

pokazowy tekst

pokazowy tekst

Rys. 2. Przykłady rodzajów znaków definiowanych programowo

Interesującym rozwiązaniem przyjętym w koncepcji programowej monitora jest zastosowanie tzw. **Window technics** (techniki okienek). Zakłada ona dynamiczny podział całego monitora na okna (ang.: windows) — p. fot. 2. Zastosowanie tej techniki w systemach aplikacyjnych pozwala analitykowi systemu na przejrzyste dzielenie bardzo ograniczonej powierzchni monitora; każde z okien może być wykorzystane do innych funkcji (np. w jednym są przedstawione informacje graficzne na temat struktury danych, w drugim prezentowane są wybrane dane z tej struktury, w trzecim — informacje na temat możliwych działań operatora systemu, czwarte może informować o konsekwencjach wyboru któregoś z działań prezentowanych w oknie trzecim — taki model zastosowano w systemie wyszukiwania informacji CALIBAN). Analityk systemu może pozwolić użytkownikowi na otwieranie, zamykanie, przesuwanie i zmianę wielkości okien. Technika ta ma decydujące znaczenie dla pracy interakcyjnej.

Dalszym krokiem w udoskonaleniu współpracy człowiek-maszyna jest zastosowanie tzw. **menu-technics.** W każdym stanie, w jakim znajduje się konkretny system aplikacyjny, możliwe jest podjęcie różnych działań. Zadaniem analityka, a później programisty systemu aplikacyjnego, jest pokazanie tych możliwości użytkownikowi systemu. Informacje te powinny być zrozumiałe na pierwszy rzut oka, powinny minimalizować możliwość pomyłki, a także powinny być adresowane, jeżeli to możliwe, zarówno do użytkownika pracującego ciągle z systemem, jak i użytkownika pracującego od czasu do czasu. Rozwiązaniem idącym w tym kierunku jest właśnie **menu-technics.** Jak konsument w restauracji, tak operator systemu ma możliwość wybrania smakowitego kąska, np. pierwszy z nich — **schabowego,** a drugi — **display.** W przeciwieństwie do menu w restauracji, menu na monitorze zawiera tylko pozycje mające sens w danym stanie systemu.

Dzięki zaprojektowaniu i zrealizowaniu powyższych środków, udało się z nawiązką zrealizować zadania związane z redagowaniem tekstów. Powstała cała rodzina edytorów. Najprostszym jest edytor tekstów używany obecnie prawie wyłącznie do redagowania programów. Umożliwia on pisanie, niszczenie, kopiowanie, przenoszenie, wyszukiwanie tekstów w ramach jednego lub większej liczby niezależnych plików znakowych (ang. textfile). Przewidziano też użycie markodefinicji, pozwalających za jednym naciśnięciem przycisku myszki lub klawiatury wybrać uprzednio zaprojektowane operacje. Operacje może projektować sam operator, ucząc system. Odbywa się to przez naciśnięcie przycisku **learn,** zademonstrowanie systemowi co ma robić przez dokładne wykonanie ciągu operacji, które ma potem samodzielnie wykonywać oraz — ponowne naciśnięcie przycisku **learn.** Wykonanie operacji nastąpi wtedy, gdy operator w dowolnym momencie pracy z edytorem naciśnie przycisk **execute.** Źródłem tej konstrukcji jest powszechnie stosowany w edytorach rozkaz **repeat.** Wysokowyspecjalizowanym edytorem jest edytor programów ukierunkowany na programowanie w MODULI-2. Występujące w nim makrodefinicje samodzielnie tworzą standardowe struktury języka, zgodnie z przyjętym układem graficznym.

Daleko szersze zadania postawiono przed edytorem tekstów zwanym ANDRA. Jego zadaniem jest umożliwienie redagowania tekstów i ich łamanie na poziomie wymaganym dla publikacji naukowych. Oprócz funkcji wykonywanych przez opisane edytory, ANDRA ma zadanie określenia kroju czcionek (rys. 2) wybranych fragmentów tekstu i sposobu rozmieszczenia poszczególnych elementów tekstu. Projektowanie tekstu odbywa się dwuetapowo. Pierwszym etapem jest zaprojektowanie możliwych postaci, jakie może przyjąć tekst lub — częściej — cały zbiór tekstów. Należy określić, jakie rodzaje czcionek mogą być użyte w tekstach, jakie odstępy mają występować pomiędzy literami, pomiędzy wierszami, jakich tabulogramów można użyć w tekście, jaki należy przyjąć sposób paginacji i — określić jeszcze wiele innych parametrów mogących charakteryzować tekst, które są chlebem codziennym redaktora technicznego gazety czy wydawnictwa książkowego. Informacje te są sformalizowane i zapamiętone w specjalnym pliku zwanym **styl** (ang. style). Redagując tekst musimy zdecydować, z jakiego stylu, czyli z jakich możliwości projektowania tekstu, będziemy korzystali. Samo redagowanie odbywa się w sposób opisany w poprzednim edytorze, natomiast elementy stylu określone są za pomocą **menu-technics.** Obecna wersja edytora umożliwia pisanie artykułów i książek odpowiadających najwyższym wymaganiom edytorskim. Istnieje tutaj parę stopni wtajemniczenia: łamanie za pomocą stylu przygotowanego przez bardziej wtajemniczonych, samodzielne projektowanie stylu, samodzielne projektowanie atrybutów stylu (np. znaków). Jest to metoda umożliwiająca korzystanie z systemu przez operatorów o różnych kwalifikacjach.

Następnym krokiem w rozwoju edytorów jest redagowanie obrazów — grafika komputerowa. Najmniejsze zadania w tym zakresie postawiono przed edytorem obrazów zwanym SIL. Jego zadaniem jest umożliwienie:
— kreślenia linii poziomych i pionowych
— kreślenia znaków z biblioteki uprzednio zaprojektowanych znaków (np. koła i strzałki, znaki takie jak diody, schematy układów scalonych pozwalające projektować układy elektroniczne)
— opatrywania ich krótkimi opisami.

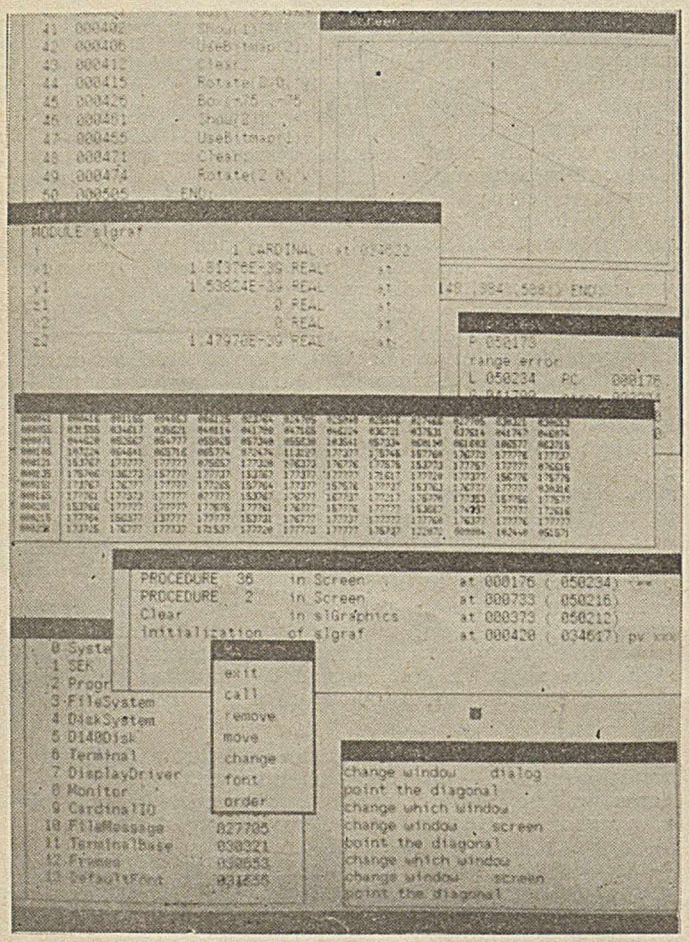
Większe wymagania postawiono przed edytorem obrazów DRAW. Za jego pomocą można kreślić dowolne krzy-

we, korzystać z biblioteki znaków, a także opisywać powstałe obrazy. Dzięki temu edytorowi można kreślić rysunki techniczne, a także rysunki zbliżone do odręcznych. Dokładność tak powstałych rysunków, w obu edytorach, jest uzależniona od zdolności rozdzielczej monitora ekranowego. Dla ułatwienia kreślenia zastosowano raster punktowy pozwalający pozycjować punkty charakterystyczne rysunków. Obrazy otrzymane za pomocą tych edytorów są zapamiętywane, jak wszystkie inne zbiory, jako ciąg bajtów.

Bardzo ważnym elementem całego przedsięwzięcia jest uzyskiwanie papierowych kopii tak pracowicie projektowanych tekstów i rysunków. Powinny one być przedstawione z rozdzielczością zbliżoną do rozdzielczości monitora (prawie pół miliona punktów), na którym je projektowano. Papier drukarski powinien odpowiadać jakości i wymiarom papieru korespondencyjnego (nie może to być typowy papier dla wydruków komputerowych). Napisano programy umożliwiające spełnienie tych warunków. Zastosowano drukarkę laserową firmy CANON. Dwie takie drukarki zaspokajają zapotrzebowanie Instytutu.

Opisane środki w pełni odpowiadają potrzebom związanym z pracami biurowymi — począwszy od informacji wewnątrzinstytutowej, a na redagowaniu książek skończywszy.

Komputer jest stosowany również do projektowania i testowania oprogramowania. O sposobach redagowania i organizacji plików wspomniano już powyżej. Spośród narzędzi wykorzystywanych przez programistę należy wymienić **debugger** (fot. 2). Jest to system umożliwiający badanie sposobu realizacji programu. Programista może, między innymi, obserwować sposób i kolejność wywoływania modułów i procedur, wartości zmiennych, zawartość pamięci. **Debugger** i inne programy w pełni zaspokajają potrzeby programisty i analityka systemów informatycznych w zakresie tworzenia, testowania oraz dokumentowania programów.



Fot. 2. „Okna" — przykład dynamicznego podziału ekranu monitora

KONFIGURACJA SPRZĘTOWA I ROZWÓJ SYSTEMU

Sprzęt komputera LILITH (fot. 1) składa się z:

— mikroprocesora segmentowego Am2901
— pamięci o pojemności 128 K słów 16-bitowych
— pamięci mikrokodu o pojemności 2K instrukcji
— sterownika pamięci dyskowej
— monitora ekranowego
— sprzęgu klawiatury
— czytnika położenia myszki
— sprzęgu V24 (maks. 9600 bodów).

Procesor pracuje z cyklem 150 ns, odpowiadającym interpretacji jednej mikroinstrukcji (najczęściej jeden rozkaz M-kodu odpowiada 2—5 mikroinstrukcjom). Moc obliczeniowa maszyny odpowiada w przybliżeniu trzem maszynom PDP 11/40.

Aby zrozumieć decyzje o wyborze procesora Am2901, należy pamiętać, że wyboru dokonywano w 1977 roku. Nie były wówczas jeszcze dostępne dzisiejsze procesory jednoukładowe, a tym bardziej nie było możliwości projektowania i produkcji procesorów w krótkich seriach.

W systemie użyto polichromatycznego monitora graficznego. Każdy punkt obrazu jest odwzorowany osobnym bitem pamięci (ang. bit-map) — zapamiętanie obrazu całego monitora zajmuje ok. 22% całej pamięci. Jest to cena za możliwość prezentacji tekstów złożonych dowolną czcionką (także — cyrylica i ideogramy) oraz — grafiki. Mimo tak dużej rozdzielczości uzyskano zadawalającą szybkość — wypełnienie całego monitora obrazem, łącznie z transformacją tekstu z kodu ASCII na odpowiadające mu obrazy w odpowiednim formacie, trwa ok. 0,25 s. Jest to wynik imponujący, jeśli się weźmie pod uwagę fakt, że stosując najmniejszą czcionkę, na monitorze można zamieścić do 10 tys. znaków.

Jako pamięć masową wybrano dysk typu D-120 firmy HONEYWEL-BULL. Są to dyski wymienne. Ma to bardzo duże znaczenie dla uelastycznienia zastosowań systemu. Wprawdzie mówimy, że LILITH jest komputerem osobistym, jednak dzięki wymienności dysków — użytkownik czuje się związany tylko ze swoimi danymi zapisanymi na dysku (ma możliwość nieograniczonego rozbudowywania swoich zasobów), który może umieścić w jednym z wielu komputerów (szczególnie istotne w przypadku awarii), a nie tylko w jednym. Ma to jeszcze większe znaczenie w sytuacji, gdy nie każdy użytkownik dysponuje swoim własnym komputerem. Wymienny dysk o dużej pojemności, zastępujący kopiowanie plików po każdej sesji z komputerem ze stałego dysku maszyny (np. Winchester) na dyski elastyczne, okazał się bardzo dobrym wyborem na etapie definiowania systemu. Dysk ma pojemność 10 M bajtów. Teoretyczna szybkość transmisji wynosi 720 KB/s, jednak efektywna szybkość pisania i czytania plików wynosi 60 KB/s. Pamięć podzielona jest na bloki po 2048 bajty.

Naturalną konsekwencją rozwoju, po zrealizowaniu zadań początkowych, było połączenie maszyn w lokalną sieć komputerową. Założono, że wszystkie węzły sieci są izomorficzne — przewidywano połączenie tylko komputerów LILITH. Sieć powinna łączyć ok. sto maszyn w odległości nie większej niż 500 metrów. Minimalna szybkość transmisji powinna wynosić jeden megabod, co jest szybkością zadowalającą nawet dla przesyłania animowanych obrazów. Wybór padł na model ETHERNET opracowany w Stanford.

Sieć nazwana MAGNET ma bardzo prosty sprzęt. Oprogramowanie umożliwia różnorodny dostęp do zasobów sieciowych, najprostszy polega na korzystaniu z tzw. **remote files.** Umożliwia on dostęp do wszystkich plików na dyskach zainstalowanych na komputerach włączonych w danym momencie do sieci. Sposób dostępu do zasobów, z punktu widzenia programisty, jest taki sam, jak by były one zgromadzone na dysku komputera na którym właśnie pracuje (w specyfikacji pliku programista musi podać nazwę urządzenia). Użytkownik sieci utożsamiany jest z dyskiem, a nie z maszyną.

Krańcowo różną metodą traktowania zasobów sieciowych jest tzw. **mail system music** (ang. most useful system for internal communication). Każdy użytkownik ma swoją **mail-box** (skrzynkę pocztową) działającą trochę inaczej niż normalna skrzynka pocztowa — można do niej wrzucić listy przeznaczone dla innych odbiorców (skrzynka pocztowa przed urzędem pocztowym), ale można z niej także

wybierać listy przesłane przez innych (skrzynka pocztowa przed domem odbiorcy). „Listem" może być dowolny plik (tekst, obraz, skompilowany program). Jak każdy normalny list, jest on zapakowany w kopertę zawierającą informację o odbiorcy i nadawcy. **Mail-system** odpowiada za obieg listów w sieci. Aby m.in. umożliwić ciągły dostęp do wszystkich skrzynek, postanowiono wyposażyć sieć w tzw. **file server** — centralną pamięć sieci. Jej zadaniem jest również pamiętanie centralnych zasobów sieci (np. całego dostępnego oprogramowania) oraz pełnienie funkcji pamięci dla drukarek laserowych. Centralną pamięcią sieci jest dysk typu M2351A firmy FUJITSU Ltd, o pojemności 474 MB i szybkości transmisji 1,86 MB/s.

* * *

Pełen opis projektu MODULA-2 LILITH jest w krótkim artykule nierealny. Celem artykułu było jedynie wskazanie paru, subiektywnie wybranych problemów. Całkowicie pominięto systemy zaprojektowane i zaimplementowane w nowym środowisku, które nie należą do jego jądra.

Pragnę podziękować prof. N. Wirthowi za liczne i konstruktywne dyskusje umożliwiające m.in. napisanie tego artykułu.

LITERATURA

[1] Abramowicz W.: Computer Mail System. Wewnętrzny raport Instytutu Informatyki Politechniki Federalnej w Zurichu, 1982
[2] Beretta G. i in.: XS-1: An Integrated Interactive System and its Kernel. W: Proceedings of the 6th International Conference on Software Engeenering. Tokyo, 1982
[3] Geissmann L.: Separate Compilation in Modula-2 and the Structure of the Modula-2 Compiler on the Personal Computer Lilith. Praca doktorska, Zurich, 1983
[4] Geissmann i in.: Lilith Handbook. A Guide for Lilith Users and Programmers. Wewnętrzny raport Instytutu Informatyki Politechniki Federalnej w Zurichu, 1982
[5] Gutknecht J.: System Programming in Modula-2: Mouse and Bitmap Display. Raport nr 56 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1983
[6] Gutknecht J., Weniger W.: Andra: The Document Preparation System of the Personal Workstation Lilith. Ukaże się w: Software — Practice and Experience
[7] Hoppe J.: A Simple Nucleus Written in Modula-2. W: Software — Practice and Experience, Vol. 10, p. 697—706, 1980
[8] Hoppe J.: Magnet. Local Network for Lilith Computer. Wewnętrzny raport Instytutu Informatyki Politechniki Federalnej w Zurichu, 1982
[9] Hoppe J.: Remote Files for the Lilith Computer. W: Proc. IFIP 83 Congress. Nort-Holland Publ., 1983
[10] Hoppe J.: A Local Area Network for the Lilith Computer. W: Proc. DECUS Europe Symposium, Zurich, 1983
[11] Jacobi Ch.: The Lilith Architecture, its Design in View of Code Generation. W: H. Langmaack i in. (ed.): Implementierung PASCAL-artiger Programmiersprachen. Teubner-Verlag, Stuttgart, 1982
[12] Jacobi Ch.: Code Generation and the Lilith Architecture. Praca doktorska, Zurich, 1983
[13] Knudsen S. E.: Medos-2: A Modula-2 Oriented Operating System for the Personel Computer Lilith. Praca doktorska, Zurich, 1983
[14] Koch J. i in.: Modula/R Report Lilith Version. Wewnętrzny raport Instytutu Informatyki Politechniki Federalnej w Zurichu, 1983
[15] Ostler F. L.: An SMD Disk Controller for Lilith Computer. Raport nr 52 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1983
[16] Rebsamen J. i in.: LIDAS — A Database System for the Personal Computer Lilith. The Database Management. Raport nr 50 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1982
[17] Rebsamen J., Zehnder C. A.: Automatische Erzeugung von konsistenzerhaltenden Transaktionen: Ein Hilfsmittel zur Datenmanipulation auf Arbeitsplatzrechnern. W: 12, Jahrestagung der Gesellschaft für Informatik, Informatik-Fachberichte 57, 595—606 Springer-Verlag, Juli 1982
[18] Wirth N.: Modula — A language for Modular Multiprogramming. W: Software — Practice and Experience, Vol. 7, 3—35, 1977
[19] Wirth N.: A Personal Computer designed for use with a high-level language. W: Remmele W., Schecher H. (ed.): Microcomputing. Teubner-Verlag, Stuttgart, 1979
[20] Wirth N.: Modula-2. Raport nr 27 i 36 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1978 i 1980
[21] Wirth N.: The Personal Computer Lilith. Raport nr 40 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1981
[22] Wirth N.: A Personal Computer for the Software Engineer. W: 5th International Conference on Software Engineering 2—16. March 9—12, 1981, San Diego, California, 1982
[23] Wirth N.: Programming in Modula-2. Springer-Verlag, 1982
[24] Zehnder C. A.: Database Techniques for Professional Workstations. Raport nr 55 Instytutu Informatyki Politechniki Federalnej w Zurichu, 1983.

KALENDARZ

Kwiecień

11—13, Paryż: STACS — konferencja na temat teoretycznych aspektów informatyki — organzator: AFCET oraz Gesellschaft für Informatik
17—19, Tuluza (Francja): VI międzynarodowe kolokwium na temat programowania — organizator: Universitè Paul Sabatier
23—25, Pitsburg (USA): konferencja na temat rozwoju oprogramowania użytkowego — organizator: ACM Sigsoft/Sigplan
25—27, Paryż: 13. międzynarodowe seminarium „Banki i komputery" — organizator: INSIG, Paryż

Maj

14—17, Amsterdam (Holandia): międzynarodowa konferencja telekomunikacyjna IEEE — organizator: TACM

Czerwiec

4—6, Nicea (Francja): kolokwium na temat predyspozycji w programowaniu — organizator: AFCET
7—8, SophiaAntipolis (Francja): konferencja na temat jakościowej oceny predyspozycji w programowaniu — organizator: AFCET
1—13, Paryż: PROLAMAT — międzynarodowa Konferencja na temat języków programowania dla obrabiarek — organizatorzy: IFIP oraz IFAC
17—21, Kopenhaga: XXVI międzynarodowa konferencja TIMS — organizator: The Institute of Management Sciences
26—29, Rzym: II światowa konferencja na temat międzynarodowego przepływu danych — organizator: Intergovernmental Bureau for Informatics

Lipiec

20—28, Montreal (Kanada): VII międzynarodowa konferencja na temat rozpoznawania obrazów — organizator: International Association for Pattern Recognition

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

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

RYSZARD KOTT
DANUTA MAGDZIK
Instytut Informatyki
Politechnika Warszawska

Techniki interpretacji dla mikrokomputerów

Wraz z rozpowszechnianiem się mikrokomputerów, a szczególnie komputerów osobistych, dużą popularnością znów zaczęły się cieszyć interpretery. Najczęstszym sposobem realizacji współczesnych systemów interpretacji dla mikrokomputerów staje się technika tzw. kodów nizanych (ang. threaded code, TC). Najlepszym przykładem możliwości tej techniki jest język, a właściwie system programowania — FORTH.

W poniższym artykule — omówimy — na tle interpretacji klasycznej — cztery odmiany kodów nizanych: podprogramowy (STC — subroutine threaded code), bezpośredni (DTC — direct threaded code), pośredni (ITC — indirect threaded code), znacznikowy (TTC — token threaded code). Pomiędzy nimi są dwie zasadnicze różnice — w sposobie reprezentacji kodu operacyjnego oraz w sposobie w jaki typ dynamicznego argumentu może wpływać na dobór operatora.

Jak wiadomo, techniki interpretacji oparte wyłącznie na interpretowanym tekście są nieefektywne. Dlatego tekst programu interpretowanego jest tłumaczony zazwyczaj na pewną postać pośrednią, tzw. kod pośredni (nie jest to kod maszynowy). Pozwala to na znaczną oszczędność zajętości pamięci i przyspieszenie interpretacji. Wszystkie omawiane w artykule techniki należą do klasy kodów pośrednich.

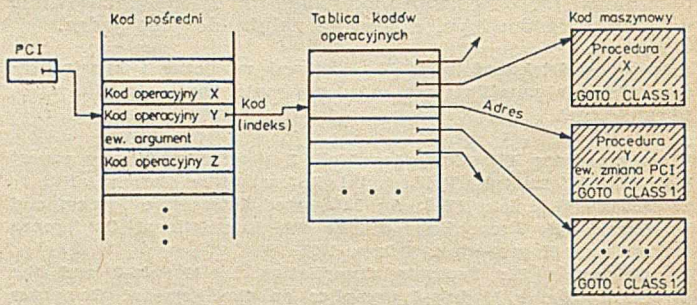
METODA KLASYCZNA

W klasycznej metodzie interpretacji kompilator generuje tylko kod pośredni. Kod programu składa się z indeksów tablicy kodów operacyjnych (TKO), gdzie są umieszczone adresy procedur interpretera, oraz z adresów używanych zmiennych. Klasyfikacja operatorów zależnych od ich typu jest przeprowadzana — jeśli to konieczne — przez jawne testowanie. Ogólny schemat interpretacji klasycznej jest następujący:

```
CLASS1  :  increment(PCI)
CLASS2  :  go to TKO[MEM[PCI]]
```

gdzie **PCI** oznacza licznik rozkazów interpretera, zaś **MEM** — pamięć. W interpreterze klasycznym, w przeciwieństwie do innych technik, pętla interpretacyjna jest jawna. W rzeczywistości krok **CLASS2** może być bardziej złożony, gdyż trudno jest oddzielić aktualny kod operacji od wartości **MEM[PCI]**. Każda procedura wykonawcza interpretera kończy się skokiem do **CLASS1**. Postać programu w kodzie pośrednim dla tego schematu interpretacji przedstawiono na rysunku 1.

W interpreterze klasycznym format instrukcji może być dowolnie zmieniany, a nawet inny dla każdej instrukcji. W konsekwencji, wobec wzrostu czasu dekodowania instrukcji, opłaca się używać kodów redukujących rozmiary programu. Dobre wyniki można uzyskać przy zastosowaniu kodów Huffmana.



Rys. 1. Postać programu interpretowanego dla klasycznej metody interpretacji (zakreskowano kod efektywny; maszynowy)

W przeciwieństwie do pozostałych czterech metod, gdzie adresy procedur interpretera są częścią instrukcji, postać instrukcji nie zależy tu od wersji interpretera. Wszystkie procedury interpretera, niezależnie od tego czy są one używane czy nie, stanowią część kombinacji: interpreter-program interpretowany.

Metoda klasyczna nie pozwala na stosowanie operacji, które zależą od typu argumentu. Znaczenie wszystkich wystąpień poszczególnych instrukcji jest takie samo i rozróżnienie typu argumentu można osiągnąć jedynie przez jawne testowanie w procedurze interpretera. W metodzie tej łatwo jest zmienić znaczenie wszystkich wystąpień poszczególnych instrukcji przez modyfikację tabeli kodów operacyjnych. W ten sposób można np. implementować śledzenie procedury.

CZTERY ODMIANY KODU NIZANEGO

Kod nizany jest techniką implementacji języka pośredniego, która organizuje sterowanie programu jako ciąg wywołań podprogramów. Kod nizany jest szczególnie przydatny do interpretacji — proces interpretacji składa się z przekazywania sterowania do procedur wskazanych przez ten kod. Wszystkie funkcje dostępne w języku pośrednim są wykonywane przez procedury, które nie są właściwą częścią kodu nizanego.

Kody nizane są szczególnie wygodne do organizacji interpretera jako maszyny stosowej, tj. takiej, w której operacje przeprowadza się nie w rejestrach, lecz na stosie roboczym. Ta technika jest stosowana np. w PASCALU, w interpreterach języków LISP i FORTH.

Wszystkie rodzaje kodu nizanego składają się ze struktury danych, która jest ciągiem niepowtarzalnych identyfikatorów procedur. Tradycyjnie, kod nizany jest ściśle związany z poziomem maszynowym i zawiera aktualne wskaźniki procedur (z kolei te procedury mogą być również podprogramami w języku pośrednim lub w kodzie maszynowym). Również niektóre zasoby procesora (w szczególności rejestry) są przekazywane do użytku interpreterowi kodu nizanego. Do implementacji kodu nizanego nie jest jednak konieczne ani adresowanie bezwzględne, ani używanie rejestrów.

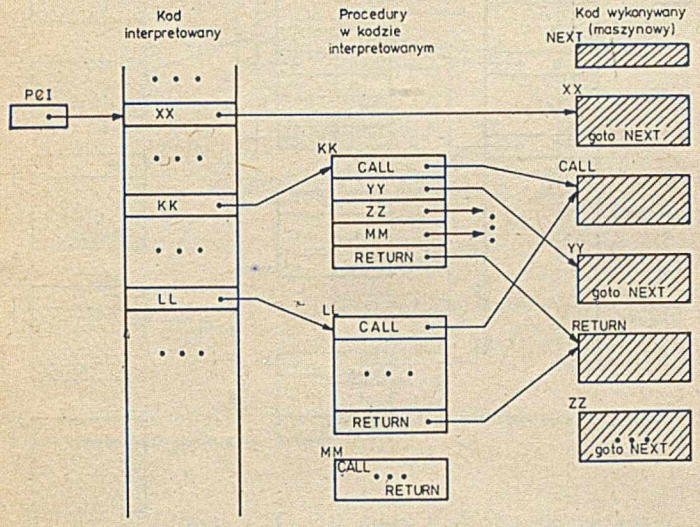
Uogólniony schemat interpretacji dla kodów nizanych można przedstawić za pomocą trzech podstawowych operacji:

Życiorys mgr. inż. RYSZARDA K. KOTTA przedstawiliśmy w nr 1 z 1981 r.

DANUTA MAGDZIK jest studentką piątego roku Politechniki Warszawskiej (Instytut Informatyki, Wydz. Elektronika). Specjalizuje się w oprogramowaniu podstawowym komputerów.

NEXT — obliczenie następnego wskaźnika do interpretacji (tą operacją musi kończyć się każda procedura w kodzie maszynowym)
ENTRY — odwołanie (wejście) do procedury niższego poziomu (w kodzie pośrednim)
RETURN — powrót z procedury w kodzie pośrednim.

Ogólny przepływ sterowania dla interpreterów z kodem nizanym przedstawiono na rysunku 2. Procedury w kodzie maszynowym są procedurami interpretera (jądro wykonawcze). Procedury w kodzie pośrednim są najczęściej procedurami użytkownika generowanymi przez kompilator, choć mogą być również procedurami interpretera. Ponadto ze zmiennymi związane są przeważnie procedury dostępu do nich.



Rys. 2. Ogólny schemat interpretacji dla kodów nizanych [1])

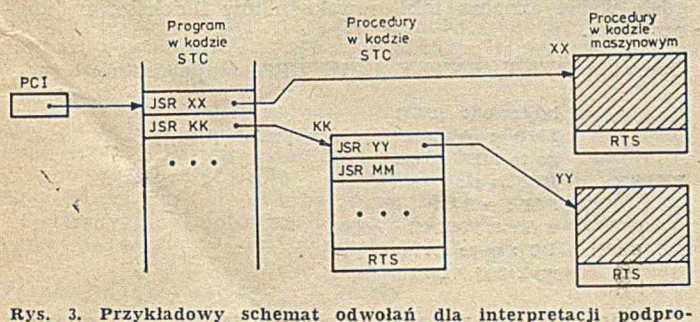
Podprogramowy kod nizany — STC

W tej metodzie kod pośredni stanowi ciąg wywołań podprogramów. Każde wywołanie składa się z pojedynczej operacji w języku pośrednim, niezależnej od komputera, w którym rezyduje. STC jest mechanizmem kontroli szeroko stosowanym na poziomie sprzętowo-maszynowym. Ogólny schemat interpretacji jest następujący:

STC1 : increment (PCI)
STC2 : wykonaj instrukcję MEM[PCI]

Operacje podstawowe mają postać:

NEXT — para instrukcji **RTS** i **JSR**
ENTRY — instrukcja **JSR** (skoku do procedury)
RETURN — instrukcja **RTS** (powrotu z procedury)
Przykład odwołań dla kodu STC przedstawiono na rysunku 3.



Rys. 3. Przykładowy schemat odwołań dla interpretacji podprogramowego kodu nizanego (STC)

STC jest najbardziej ogólnym językiem pośrednim, gdyż nie używa wprost języka maszynowego. Programiści rzadko piszą programy składające się jedynie z wywołań podprogramów, ale jest to czasem produkt wyjściowy kompilatora. STC wnosi mniejszy narzut wykonania niż większość języków pośrednich, ponieważ jego interpreter posługuje się bardziej sprzętem niż ciągiem instrukcji. STC może być optymalizowany przez zapisanie wewnątrz niego operacji w kodzie maszynowym będących zbyt dużym obciążeniem dla programu. Oczywiście, tak otrzymany zoptymalizowany kod nie jest niezależny od maszyny.

[1]) Na rysunkach 2, 4, 5 i 6 nazwa CALL oznacza operację ENTRY lub adres procedury ENTRY

Bezpośredni kod nizany — DTC

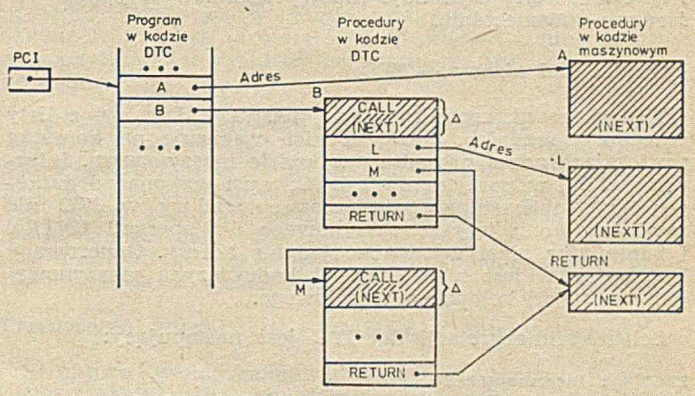
DTC składa się z ciągu wywołań procedur w języku maszynowym, w których pominięto kod wywołania. Ma on więc postać listy adresów, z których każdy wskazuje procedurę. Ponieważ DTC nie zawiera żadnego kodu operacji, konieczne jest napisanie krótkiego programu w języku maszynowym, który czyta następny adres z listy i przekazuje tam sterowanie. Słowo „bezpośredni" w nazwie kodu oznacza, że pod wskazanym adresem oczekuje się procedury w kodzie maszynowym, a nie w kodzie pośrednim, Podstawowy schemat interpretacji dla DTC jest następujący:

DTC1 : increment(PCI)
DTC2 : wykonaj procedurę spod adresu MEM[PCI]

Natomiast podstawowe operacje interpretera mają postać:

NEXT: **Increment (PCI)**
TMP:=MEM [PCI]
goto TMP
ENTRY: **PCI na stos powrotów**
PCI := TMP+(Δ —1)
wykonaj NEXT
RETURN: **PCI := ze stosu powrotów**
wykonaj NEXT

Schemat odwołań dla interpretacji kodu DTC przedstawiono na rysunku 4.



Rys. 4. Przykładowy schemat odwołań dla bezpośredniego kodu nizanego (DTC)

W niektórych komputerach wszystkie czynności związane z operacją **NEXT** można wykonać za pomocą jednego rozkazu. W takim przypadku ostatnią czynnością przy wykonywaniu operacji **ENTRY** i **RETURN** jest wykonanie takiego rozkazu. Pętla interpretacji jest więc niejawna i rozsiana po całym interpreterze. Podobnie jest dla innych kodów nizanych, ale w tym przypadku lepiej widać możliwość zwiększenia szybkości interpretacji. W szczególności kod DTC idealnie nadaje się dla komputera PDP-11, gdyż może być na nim bardzo sprawnie zrealizowany dzięki instrukcji **JUMP @(r)+** (skok indeksowany pośredni z autoinkrementacją rejestru), gdzie **r** jest jednym z rejestrów PDP-11 pełniącym funkcję licznika rozkazów interpretera, tj. **PCI.**

Operacja **ENTRY** nie musi być wstawiana w całości do procedury pośredniej, jak to wynika z rysunku. Można wyodrębnić wspólny fragment operacji, a w kodzie pośrednim znajdzie się wówczas tylko skok (**JMP**) do tego fragmentu, tj. zazwyczaj trzy bajty. Nie może to być oczywiście wywołanie podprogramu. (Przy opisie ENTRY długość Δ-1 podano w słowach; w bajtach byłoby Δ-2; w najczęstszym przypadku **PCI=TMP+1 (bajt)**).

W przypadku DTC kompilator generuje nie tylko program w kodzie pośrednim, ale również procedury dostępu

do zmiennych (w kodzie pośrednim albo maszynowym), rozmieszcza też w pamięci same zmienne. Jeśli w programie używa się np. zmiennych **X** i **Y**, to generowane są także procedury umieszczania tych zmiennych na stosie:

```
PUSHX :  TMP : = X
         goto    PUSH
PUSHY :  TMP : = Y
         goto    PUSH
```

gdzie **X,Y** oznaczają adresy zmiennych **X** i **Y**, zaś **PUSH** jest stałą procedurą interpretera zakończoną **NEXT.**

Format instrukcji DTC jest określony i składa się z adresów. Niekiedy częścią programu w DTC są argumenty bezpośrednie, które mogą być pobrane przez **PCI.** Aktualna instrukcja odpowiada za to, aby **PCI** otrzymał właściwą wartość.

Instrukcje zależą od szczególnej wersji interpretera (np. od adresów procedur interpretera). Konsolidator zamienia program w kodzie nizanym w ciąg procedur interpretera. W większości systemów operacyjnych konsolidator tworzy powiązane między sobą kopie modułów, co ma następujące konsekwencje:

— program w kodzie pośrednim jest zmuszony do kopiowania procedur interpretera; może to być przyczyną rozmnażania się nieaktualnych lub błędnych kopii (w tym celu należy zarezerwować duży obszar pamięci dyskowej)
— do programu interpretowanego są dodawane jedynie używane procedury.

DTC nie jest przystosowany do wykonywania operacji zależnych od typu argumentu, ale ustalony format instrukcji umożliwia łatwe dodanie nowych instrukcji.

DTC jest bardziej efektywny, gdy używa się specjalnych procedur pobierania zmiennych z pamięci i zapisywania ich do pamięci. Takie procedury są specyficzne dla każdego programu interpretowanego i nie mogą być częścią interpretera, który wykonuje kilka programów w języku pośrednim równocześnie.

Pośredni kod nizany — ITC

Kod pośredni, zgodnie ze swą nazwą, składa się z listy adresów pośrednich. Każdy z nich wskazuje na komórkę zawierającą adres procedury w kodzie maszynowym. Interpreter ITC, w porównaniu z DTC, musi wykonać dodatkowy krok obliczeniowy, natomiast procedury w ITC nie zawierają kodu języka maszynowego dla operacji **ENTRY** i kompilator musi wytwarzać jedynie adresy. Generowany kod pośredni jest niezależny od kodu języka maszynowego, a więc od komputera docelowego.

Schemat interpretacji dla ITC jest następujący:

```
ITC1  : increment(PCI)
ITC2  : wykonaj procedurę spod adresu MEM [MEM [PCI]]
```

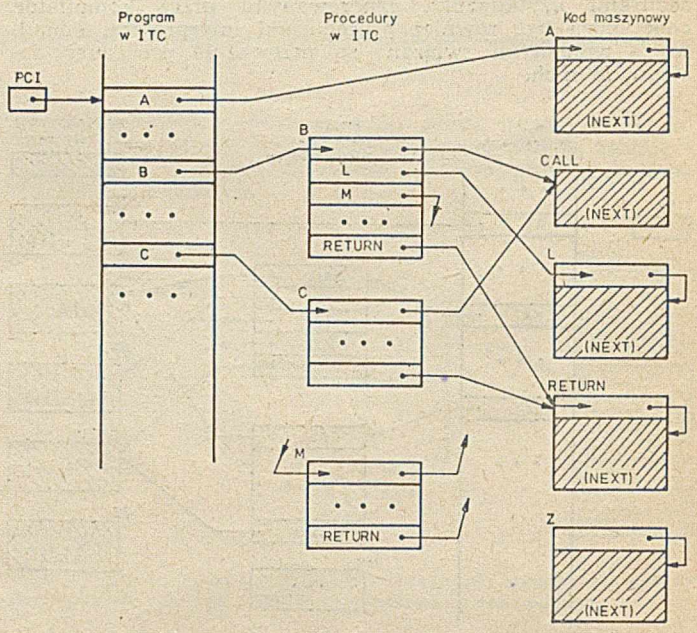
Natomiast operacje **NEXT, ENTRY** i **RETURN** mają postać:

```
NEXT:    increment (PCI)
         TMP := MEM [PCI]
         goto MEM [TMP]
ENTRY:   PCI na stos
         PCI := TMP
         wykonaj NEXT
RETURN:  PCI := ze stosu
         wykonaj NEXT
```

Przykład odwołań dla kodu pośredniego przedstawiono na rysunku 5.

Format instrukcji jest określony. Argumenty bezpośrednie są rzadko stosowane, gdyż nawet stałe mogą być traktowane tak samo jak zmienne, bez potrzeby wprowadzania dodatkowych procedur dostępu. Dla utworzenia kodu ITC wystarczy wygenerować tylko adresy (w metodzie DTC należy wytworzyć adresy i kod maszynowy). Dowodzi to przenośności generatora kodu dla kodu ITC. Przykładowo, używający ITC system MACRO SPITBOL zaimplementowano w bardzo krótkim czasie na blisko 20 różnych komputerach.

ITC skupia operacje, które zależą od typu argumentu. Ma to tę zaletę, że specjalne przypadki (śledzenie wartości zmiennych, połączenie wejścia-wyjścia) mogą być traktowane bez dodatkowego narzutu związanego z ogólnością używanych procedur. Trudna jest modyfikacja wszystkich wystąpień określonej instrukcji, co nie stwarza kłopotów w przypadku interpretacji klasycznej. Łatwo można zmieniać odwołania do procedur, które mają specjalny poziom pośredności w punkcie wejścia (tak jak w procedurze B), ale jeśli procedury nie mają takiego punktu, to w celu zmiany odwołania do tych procedur trzeba przejrzeć cały interpretowany program.



Rys. 5. Przykładowy schemat odwołań dla pośredniego kodu nizanego (ITC)

Znacznikowy kod nizany — TTC

Przedstawione dotąd rodzaje kodu nizanego używają wskaźników, które są aktualnymi adresami procedur w pamięci. Używając adresów pamięci dla dostępu do procedur, traci się pamięć, gdy liczba procedur w systemie jest dużo mniejsza niż przestrzeń adresowa. Dzięki użyciu krótkich znaczników dla identyfikacji podprogramów można znacznie zmniejszyć rozmiar programu w języku pośrednim. Typowy TTC może być zaimplementowany przez użycie znaczników jako indeksów w tablicy adresów podprogramów.

Schemat interpretacji dla TTC jest następujący:

```
TTC1 : increment(PCI)
TTC2 : wykonaj procedurę spod adresu MEM [Table [MEM [PCI]]
```

gdzie **Table** oznacza tablicę wskaźników indeksowaną znacznikami.

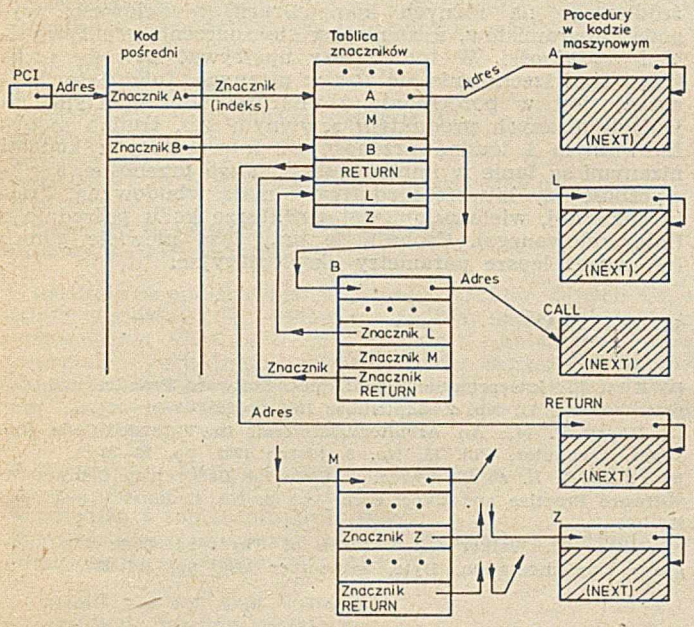
Operacje **NEXT, ENTRY** i **RETURN** mają postać:

```
NEXT:    increment (PCI)
         TMP : = MEM [Table [MEM [PCI]]]
         go to MEM [TMP]
ENTRY:   PCI na stos
         PCI : = TMP
         wykonaj NEXT
RETURN:  PC : = ze stosu
         wykonaj NEXT
```

Z zapisu wynika, że adres wzięty z tablicy jest adresem pośrednim. Często zaznacza się to w nazwie, mówiąc o pośrednim znacznikowym kodzie nizanym (ITTC, Indirect TTC). Podstawowe elementy interpretera z kodem znacznikowym przedstawiono na rysunku 6.

Kod znacznikowy występuje w rozmaitych odmianach. Powyżej omówiono jedynie najprostszą. Natomiast ważnym rodzajem kodu ITTC jest taki, dla którego kod pośredni składa się wyłącznie ze znaczników, a wszystkie procedury rozpoczynają się od znacznika, a nie od adresu. Postać operacji **NEXT** jest w tym przypadku następująca:

```
NEXT:
        increment (PCI)
        TMP : = MEM [Table [MEM [PCI]]]
        goto MEM[Table[MEM [TMP]]] + 1
```



Rys. 6. Przykładowy schemat odwołań dla interpretera ze znacznikowym kodem nizanym (ITTC)

Format instrukcji TTC jest stały i składa się ze znaczników będących indeksami tablicy, które są krótsze niż właściwe adresy. Kod TTC jest przenośny, a dla jego utworzenia wystarczy wygenerować znaczniki.

PORÓWNANIE RÓŻNYCH TECHNIK INTERPRETACJI

Tradycyjne implementacje interpreterów kodu nizanego miały co najmniej jeden rejestr przeznaczony do wyłącznego użytku interpretera. Implementacje na mikrokomputerach używają wszystkich zasobów mikroprocesora. Jedyna trudność w tych implementacjach polega na tym, że wszystkie procedury w języku maszynowym (gdzie są wykonywane właściwe obliczenia) muszą ochraniać rejestry procesora przed modyfikacjami i odtwarzać ich zawartość przed powrotem do interpretera. Korzystanie z zasobów maszyny warunkuje użycie standardowych podprogramów w języku maszynowym, które przesyłają parametry pomiędzy rejestrami. Muszą też istnieć procedury przesyłające wartości danych do i z rejestrów.

Można wyeliminować użycie zasobów procesora w języku pośrednim przez zapisanie rejestrów interpretera w pamięci, co pozwala na użycie procesora przez kod maszynowy. Gdy rejestry języka pośredniego są chronione na stosie, język powinien być niezależny od programowego adresowania pamięci.

Innym sposobem eliminacji użycia zasobów procesora, równie dobrym jak maksymalizacja przesłań, jest użycie kodu STC. Kod ten korzysta jedynie z licznika rozkazów i stosu adresów powrotu z podprogramu, zasobów zawsze oddawanych do sterowania przepływem programu. Tak więc tradycyjnie dostępne programiście zasoby procesorowe pozostają wolne od użycia przez kod maszynowy.

Parametry eksploatacyjne i koszty

Zasadnicze parametry eksploatacyjne, tj. szybkość interpretacji i zajętość pamięci, są dość trudne do oszacowania (por. [1]). Mają na nie wpływ takie czynniki, jak sposób realizacji **PCI** (w pamięci albo w rejestrze), czy liczba poziomów procedur w kodzie pośrednim (głębokość wywołania, przy której dociera się do kodu maszynowego).

Co do zajętości pamięci, to zważywszy bajtową strukturę pamięci mikrokomputerów oraz możliwość odpowiedniego dobrania kodów operacji, należy się spodziewać, że technika klasyczna oraz ITTC dają najbardziej zwarty kod. Dla pozostałych technik kod pośredni musi zawierać całe adresy (dwa bajty), natomiast indeksy tablic w wymienionych metodach mogą być jednobajtowe. Ponadto DTC zajmuje średnio jeden bajt więcej na procedurę (kod rozkazu **IMP**) niż ITC, a w metodzie STC kod pośredni składa się z trzybajtowych wywołań procedur.

Zakładając jeden poziom procedur dla technik nizanych, można stwierdzić, że najszybszą techniką jest DTC, szczególnie gdy licznik rozkazów interpretera (**PCI**) jest przechowywany w rejestrze. W porównaniu ze schematem klasycznym zyski biorą się z możliwości „rozsiania" sterowania interpretera. Ich źródło leży w tym, że kody nizane DTC, ITC, ITTC mają sterowanie zorganizowane jako ciąg skoków przez odpowiednio przygotowane adresy, a nie jako ciąg wywołań procedur (tj. nie zapamiętuje się adresu powrotu).

Warto zaznaczyć, że większość mikroprocesorów 16-bitowych ma tryby adresowania ułatwiające taką implementację DTC (np. INTEL 8086 i MOTOROLA 68000 mają pośredni tryb skoków, indeksowany przez rejestr, podobny do trybu skoków PDP-11). Jeśli umieścić **PCI** w pamięci, to szybkości wykonania dla DTC i metody klasycznej będą zbliżone.

Powyższe rozważania należy zmodyfikować, gdy założymy, że w pośrednim kodzie nizanym istnieje wiele poziomów procedur, zanim osiąga się kod maszynowy. Jeśli interpreter wielopoziomowy realizuje te same funkcje co jednopoziomowy, to oczywiście — jest wolniejszy. Wiąże się to z przełączaniem poziomów, tj. wielokrotnym wykonywaniem operacji **ENTRY** i **RETURN**. Im bardziej te operacje są złożone (ITC, ITTC) tym bardziej są czasochłonne. Ponieważ metoda klasyczna jest jednopoziomowa, wydawałoby się, że jest bezkonkurencyjna. Jednak, po pierwsze — implementacja **PCI** w rejestrze dla kodu DTC może dać duże zyski czasowe, niwelowane dopiero przez sporą liczbę poziomów; po drugie — stworzenie interpretera jednopoziomowego wymaga zakodowania wszystkich procedur (operacji) w języku mikrokomputera, co jest zawsze trudniejsze (droższe) i daje produkt mniej podatny na modyfikacje, a niekiedy po prostu nie jest możliwe albo opłacalne.

Wielopoziomowa struktura kodu pośredniego ma też duży wpływ na zajętość pamięci — dla dużych programów w kodzie pośrednim (a także samego interpretera) może ona być znacznie zredukowana. Jest to efekt używania procedur innych poziomów jako podprogramów. Tak więc zajętość pamięci dla kodów nizanych może być mniejsza niż dla interpretera klasycznego. W przypadku kodów nizanych największe możliwości ma ITTC.

Przenośność interpretera i programów interpretowanych

Struktura wielopoziomowa wyraźnie różnicuje wymienione techniki, gdy rozważamy przenośność wytworzonego oprogramowania (interpretera i programów użytkowych). Przy przenoszeniu oprogramowania wszystkie instrukcje w kodzie maszynowym muszą zostać ręcznie zastąpione instrukcjami nowej maszyny. Natomiast nowy kod pośredni można uzyskać dokonując ponownej kompilacji wstępnej kodu źródłowego. Jeśli ponadto kompilator jest zrealizowany za pomocą procedur interpretera w kodzie pośrednim, to można go używać od razu albo po minimalnych zmianach (np. zmiana kodu **JMP** w operacji **ENTRY** dla **DTC**). Przeniesienie kompilatora wstępnego w takiej sytuacji nie powoduje żadnych dodatkowych kosztów, gdyż wspomniane modyfikacje są identyczne dla całego oprogramowania interpretowanego.

Ponieważ przy strukturze wielopoziomowej kod maszynowy może stanowić niewielki fragment całości interpretera, to dla kodów nizanych przenoszenie oprogramowania jest bardzo ułatwione, tzn. tańsze i szybsze. Natomiast dla interpretera klasycznego kod maszynowy stanowi praktycznie 100% całości. Dla kodów ITC i ITTC kod maszynowy nie występuje w ogóle poza najniższym poziomem interpretera, w szczególności kod interpretowany w ogóle nie zależy od maszyny.

Wybór techniki

Różne postacie kodu nizanego mają różne możliwości pod względem szybkości i wydajności kodu. Najszybszą techniką jest DTC, natomiast najbardziej oszczędną pamięciowo — ITTC. Szybkość interpreterów z kodem DTC może

(ale nie musi) być większa od szybkości interpretera klasycznego, podobnie interpretery z kodem nizanym mogą (ale nie muszą) dawać bardziej zwarty kod. Gdy zwiększa się liczba poziomów, maleje szybkość, ale zmniejsza się także długość kodu. Wydaje się, że jako rozsądny kompromis pomiędzy sprzecznymi kryteriami można zalecić DTC — technikę szybką i — przy strukturze wielopoziomowej — dającą zwarty kod.

Skoro technika kodów nizanych nie ma wyraźnej przewagi co do szybkości interpretacji czy wydajności kodu (a nawet może implikować gorsze parametry niż technika klasyczna), to dlaczego przedkładamy interpretery z kodami nizanymi nad interpretery o organizacji klasycznej? Jest tak, ponieważ przy zachowaniu podobnych parametrów eksploatacyjnych, za pomocą kodów nizanych możemy otrzymać produkt bez porównania bardziej elastyczny i podatny na modyfiakcje, a więc oprogramowanie znacznie łatwiejsze do konserwacji i rozszerzania.

System taki można bardzo łatwo modyfikować. Aby dodać nową operację do interpretera z kodem nizanym wystarczy zdefiniować nową procedurę w kodzie pośrednim za pomocą translatora wstępnego, będącego częścią interpretera. Od chwili wprowadzenia do systemu, możemy posługiwać się nową operacją tak jak każdą inną już istniejącą operacją (procedurą). Przy tym — z wyjątkiem interpretera z ITTC — nie trzeba nic zmieniać w dotychczasowym interpreterze. W ten sposób można tworzyć nowe operacje interpretera o coraz bardziej złożonych funkcjach. Taka jest właśnie zasada programowania w systemie FORTH.

Obecnie zwiększa się zainteresowanie różnymi technikami interpretacji — ponieważ wraz z pojawieniem się dużej liczby mikroprocesorów wzrasta znaczenie takich cech oprogramowania, jak możliwość łatwej modyfikacji i rozszerzalności, elastyczność, łatwość przenoszenia na inny komputer, zdolność wykonywania tego samego programu źródłowego na różnych komputerach, modułowość, wygoda programistów, a zmniejsza się znaczenie efektywności i wydajności. W tym należy upatrywać powody szybkiego upowszechniania się kodów nizanych. Interpreter klasyczny jest w porównaniu z interpreterami opartymi na kodach nizanych produktem sztywnym tzn. trudno modyfikowalnym i trudno przenośnym. Interpretery z kodami nizanymi są tanie w implementacji, łatwo przenośne, a elastyczność czy łatwość modyfikacji mają wbudowaną, dzięki założonej, wielopoziomowej strukturze kodu pośredniego (interpretowanego). Cechują je przy tym niewiele gorsze lub nawet lepsze parametry eksploatacyjne.

LITERATURA

[1] Klint P.: Interpretation Techniques. Software Practice and Experience, vol. 11, No. 9, September 1981 pp. 963—973

[2] Kogge P. M.: An Architectural Trail to Threaded-Code Systems. Computer, Vol. 15, No. 3, March 1982, pp. 22—32

[3] Philips J. B. et al.: Threaded Code for Laboratory Computers. Software Practice and Experience, Vol. 8, No. 1, January 1978, pp. 277—279

[4] Ritter T., Walker G.: Varietes of Threaded Code for Language Implementation. Byte, September 1980, pp. 206—227.

MAREK SOBCZYK
ANDRZEJ SZAŁAS
Warszawa

PROLOG (2)

Poprzedni artykuł o PROLOGU zawierał intuicyjne wprowadzenie do tego języka. Przytoczymy teraz kilka przykładów, które — jak sądzimy — pozwolą Czytelnikom bliżej poznać jego mechanizmy.

Zacznijmy od programu zdolnego tworzyć „mutanty". Z dwóch zwierząt (reprezentowanych przez ich nazwy) może powstać mutant, jeśli koniec nazwy pierwszego z nich jest identyczny z początkiem nazwy drugiego. Interesującą cechą programu jest użycie w nim tej samej relacji **conc** na dwa różne sposoby: z jednej strony — by łączyć dwie listy, z drugiej zaś — by rozłożyć listę na dwie podlisty. W regule **exp** występuje standardowy predykat **exm (c)**, który powoduje wypisanie łańcucha **c** (z pominięciem cudzysłowów), przy czym długość tekstu nie powinna przekroczyć długości linii urządzenia wyjściowego:

```
„MUTANTY"

mutant(z)→ zwierzę(x) zwierzę(y) conc (a,b,x)
        dif(b,nil)  conc(b,c,y)  dif(c,nil)conc(x,c,z);
conc(nil,y,y) → ;
conc (e.x,y,e.z) → conc(x,y,z) ;
gotowy—mutant → mutant(z) exp(z) ;
exp(nil) → ;
exp(a.l) → exm(a) exp(l) ;
zwierzę("w"."i"."e"."l"."o"."r"."y" "b".nil) → ;
zwierzę("r"."y"."b"."a".nil) → ;
zwierzę("k"."u"."r"."a".nil) → ;
zwierzę("r"."a"."k".nil) → ;
zwierzę("m"."a"."l"."p"."a".nil) → ;
zwierzę("p"."a"."t"."y"."c"."z"."a"."k".nil) → ;
zwierzę("k"."o"."n"."d"."o"."r".nil) → ;
zwierzę("o"."r"."z"."e"."ł".nil) → ;
zwierzę("k"."o"."t".nil) → ;
zwierzę("t"."y"."g"."r"."y"."s".nil) → ;
zwierzę("z"."e"."b"."r"."a ".nil) → ;
```

Program ten po wywołaniu **gotowy-mutant;**, stworzy następujące mutanty: wieloryba, kurak, rakura, rakondor, rakot, małpatyczak, patyczakura, patyczakondor, patyczakot, kondoryba, kondorak, kondorzeł, kotygrys, zebrak.

Prześledźmy kolejny przykład, ilustrujący PROLOGOWĄ realizację klasycznej procedury sortowania **quicksort.** Załóżmy, że dany jest predykat **mniejsze(x,y),** definiujący porządek w jakim sortujemy elementy. Wprowadźmy procedurę **podziel(h,t,l,m),** która rozkłada listę **t** na listy **l** i **m** w ten sposób, że **l** zawiera wszystkie elementy z listy **t,** które są mniejsze od **h,** zaś **m** — wszystkie elementy z **t** większe od **h:**

```
podziel(h, a.x, a.y, z) → mniejsze (a,h) podziel (h,x,y,z) ;
podziel(h, a.x, y, a.z) → mniejsze (h,a) podziel (h,x,y,z) ;
podziel(q,nil.nil,nil) → ;
```

Teraz możemy napisać procedurę **quicksort,** wykorzystującą procedurę **podziel** oraz predykat **conc** z poprzedniego przykładu:

```
quicksort(h.t , s) → podziel(h,t,a,b) quicksort(a,a1)
                     quicksort(b,b1) conc(a1, h.b1, s) ;
```

Po wywołaniu: **quicksort(l1, l2),** lista l2 będzie zawierała posortowane elementy listy l1. Inne przykłady procedur sortowania znaleźć można w książkach [1, 4].

Przyjrzyjmy się teraz, w jaki sposób można w PROLOGU definiować struktury danych bardziej skomplikowane niż lista. Załóżmy, że mamy do rozwiązania problem, w którym wygodnie jest skorzystać z pojęcia zbioru skończonego i z typowych operacji związanych ze zbiorami

Zbiór reprezentować będziemy za pomocą listy. Wprowadzimy następujące predykaty:

element(e,s), który jest spełniony, jeśli **e** jest elementem zbioru **s**
podzbiór(s,t), stwierdzający, czy **s** jest podzbiorem **t**
przecięcie(r,s,t), powodujący, że **t** przyjmuje wartość **r∩s**
suma(r,s,t), powodujący, że **t** przyjmuje wartość **r∪s**:

```
element(e, e.p) → ;
element(e, q.p) → element(e, p) ;
podzbiór(a.x, y) → element(a,y) podzbiór(x,y) ;
podzbiór(nil, y) → ;
przecięcie(nil,x,nil) → ;
przecięcie(x.r , y , x.z) → element(x,y) / przecięcie(r,y,z) ;
suma(nil,x,x) → ;
suma(x.r, y , z) → element(x,y) / suma(r,y,z) ;
suma(x.r, y , x.z) → suma(r,y,z) ;
```

Bardziej skomplikowane przykłady struktur danych Czytelnik znajdzie w [4]. Omówione są tam także interesujące zastosowania PROLOGU. My natomiast, aby uściślić nasze poprzednie rozważania, opiszemy teraz semantykę operacyjną PROLOGU za pomocą abstrakcyjnej maszyny zwanej „zegarem PROLOGU" (por. [2]). Maszyna ta składa się z następujących elementów:

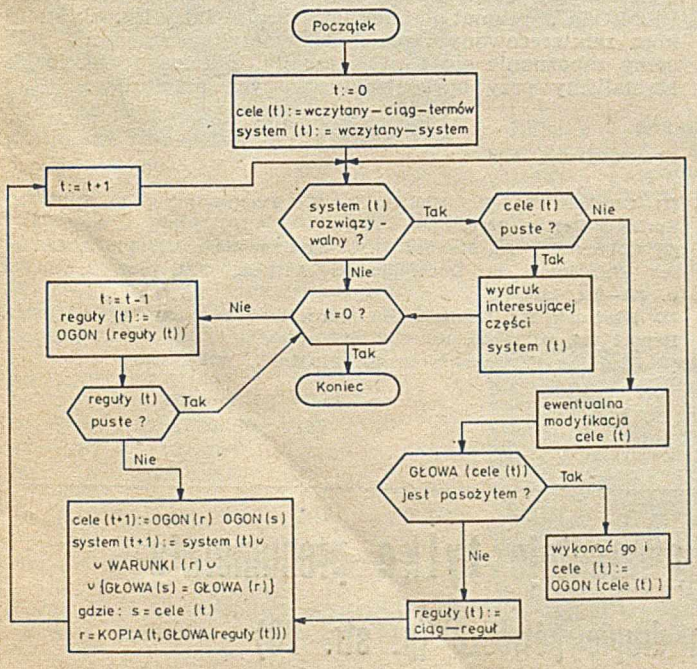
- komórki o nazwie **ciąg-reguł**, zawierającej reguły tworzące program w PROLOGU
- komórki **t** reprezentującej „czas"
- trzech nieograniczonych zbiorów komórek wskazywanych przez niujemne liczby całkowite **i**:

— **cele(i)** zawiera ciąg termów u_i
— **system(i)** zawiera system s_i
— **reguły(i)** zawiera ciąg reguł aktywnych w chwili **i**.

Jedynie komórka **ciąg-reguł** ma wartość początkową, określającą środowisko programowe w momencie rozpoczęcia interpretacji PROLOGU. Maszyna czyta rozkazy ze swej jednostki wejściowej, oraz kolejno je wykonuje, drukując za każdym razem obliczone wyniki. Postać syntaktyczna rozkazu jest następująca:

```
<rozkaz> : := <ciąg termów> ;
         : := <ciąg termów> , <system> ;
```

Oczywiście brak w rozkazie części opisywanej przez **<system>** powoduje przyjęcie systemu pustego jako wartości domyślnej. Funkcjonowanie maszyny przedstawione jest schematycznie na rysunku 1, na którym:
GŁOWA(x) — reprezentuje pierwszy element ciągu **x**, czyli lewy składnik reguły **x**
OGON(x) — reprezentuje ciąg **x** bez pierwszego elementu
WARUNKI(r) — reprezentują system równości i nierówności związanych z regułą **x**



Rys. 1. Schemat zegara PROLOGU

KOPIA(t,r) — reprezentuje kopię reguły **r** uzyskaną w chwili **t**; kopia ta nie ma wspólnych zmiennych ani z **cele(t)**, ani z **system(t)**.

Wyjaśnijmy teraz pojawiające się na rysunku pojęcie pasożyta (franc. parasite). Nazwę tę zapożyczyliśmy z [2], bowiem wydaje nam się ona adekwatna do określenia predykatów „psujących" możliwości statycznej interpretacji reguł, niezbędnych jednak w praktycznym programowaniu. Pasożytami są te predykaty standardowe, których zastosowanie wymaga znajomości przebiegu obliczeń w progaramach PROLOGOWYCH. Umożliwiają one:

— sterowanie przebiegiem programu
— modyfikowanie bieżącego zbioru reguł zawartych w **ciąg-reguł**, a więc wprowadzanie i zmiany programów
— uzyskiwanie dostępu do klasycznych funkcji arytmetycznych i operacji działających na łańcuchach
— sterowanie wejściem i wyjściem.

Sterowanie realizowane jest za pomocą pasożyta "/" i reguł predefiniowanych, które opiszemy poniżej. Identyfikatory pasożytów ujmować będziemy w apostrofy.

Pasożyt "/"

Jak już zauważyliśmy wcześniej, w programach PROLOGOWYCH często konieczna jest możliwość usunięcia wszystkich punktów wyboru, które pojawiły się pomiędzy chwilą **t-k** i chwilą bieżącą **t**. Wykonanie pasożyta / polega na przypisaniu pustego ciągu reguł do wszystkich komórek **reguła(i) dla t-k≤i≤t**. Liczba całkowita **t-k** określa ostatni moment, w którym komórka **cele(t-k)** nie zawiera rozpatrywanego wystąpienia /.

Reguła wolne(x)→'wolne'; (franc. libre)

Wykonanie pasożyta **'wolne'** nie ma żadnego efektu, jeśli **x** jest zmienną, z którą nie związano dotychczas żadnej konkretnej wartości, w przeciwnym przypadku powoduje, że **system(t)** jest nierozwiązywalny.

Reguła związane(x)→'związane'; (franc. pris)

Wykonanie pasożyta **'związane'** jest dualne do wykonania poprzedniej reguły, tzn. nie ma żadnego efektu, jeśli **x** nie jest **'wolne'** a w przeciwnym przypadku czyni **system(t)** nierozwiązywalnym.

Reguła zawieś(x,p)→'w oczekiwaniu'x p; (franc. geler)

W programach PROLOGOWYCH możliwa jest zmiana porządku elementów ciągu **cele(t)**, a więc — w szczególności — opóźnienie przepisania konkretnego elementu **p** tego ciągu aż do momentu, gdy pewna zmienna stanie się związana. Efekt ten można uzyskać przy użyciu reguły **zawieś**.

Z zasady pasożyt **'w oczekiwaniu'** nigdy nie jest wykonywany. Jest on brany pod uwagę przez operację modyfikacji ciągu **cele(t)** w zegarze PROLOGU (por. rys. 1). Zmiana dokonuje się w dwóch etapach:

- wszystkie trójki postaci **'w oczekiwaniu' x p**, które występują w ciągu **cele(t)**, są przenoszone na koniec **cele(t)**
- następnie wszystkie **p** występujące w trójkach postaci **'w oczekiwaniu' x p**, w których zmienna stała związana w chwili **t**, są przenoszone na początek ciągu **cele(t)**, zaś odpowiednie wystąpienia **'w oczekiwaniu'** oraz **x** są usuwane.

Zauważmy, że reguła **zawieś** umożliwia używanie techniki współprogramów (omówienie współprogramów w PROLOGU Czytelnik może znaleźć np. w [3]).

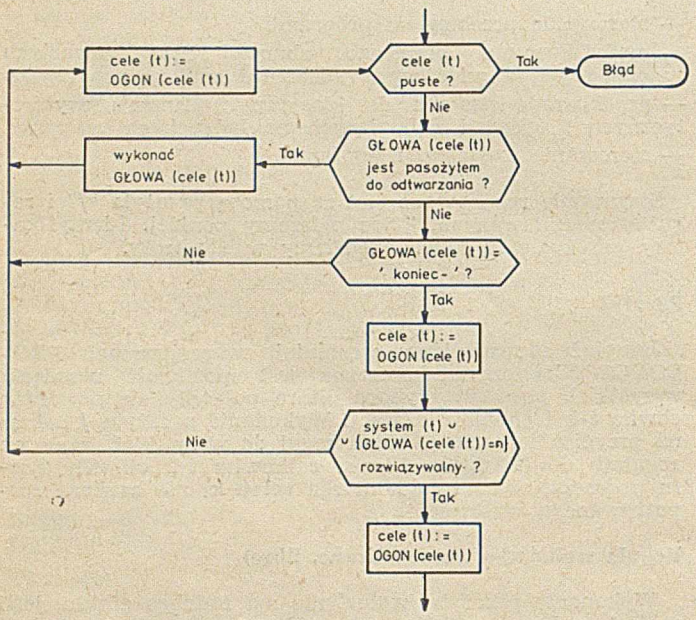
Reguła blok(n,p) → p'etykieta'n; (franc. bloc)

Reguła ta pozwala na umieszczenie etykiety **n** we wnętrzu ciągu **cele(t)**, umożliwia zatem przerwanie przepisywania pewnej liczby termów poprzez skok do tej etykiety. Rekurencyjny sposób wprowadzania etykiet tworzy z ciągu **cele(t)** strukturę bloków noszących nazwy odpowiadających im etykiet. Skok do etykiety **n** sprowadza się zatem do zakończenia wycierania bloku noszącego na-

żwę **n**. Wykonanie pasożyta **'etykieta'** daje w efekcie wyeliminowanie z ciągu **cele(t)** termu, który jest „etykietowany" tym pasożytem.

Reguła koniec-bloku(n)→ 'koniec-bloku' ; (franc. fin-bloc)

Reguła ta pozwala na bezwzględne zakończenie wycierania pierwszego wyłączonego bloku o nazwie **n**. Niektóre pasożyty nie mogą zostać jednak zignorowane w takim przypadku (tzw. pasożyty odtwarzania) i są one mimo wszystko wykonywane (rys. 2).



Rys. 2. Wykonanie pasożyta 'koniec bloku'

Reguła p.q→pq :

Pozwala ona na połączenie ciągu celów w jeden cel.

Reguły eq(x,x)→ ; oraz dif(x,y)→ , {x # y} ;

Definiują równości i różności w formułach.

W PROLOGU możliwe jest w każdej chwili modyfikowanie zawartości komórki **ciąg-reguł** poprzez dołączanie lub usuwanie pewnych reguł. Modyfikacje te dokonywane są zawsze w odniesieniu do bieżącego ciągu reguł, stanowiącego program PROLOGOWY.

Wprowadzanie programów odbywa się za pomocą predefiniowanej reguły: **wprowadź→ 'wprowadź'** ; (franc. insérer). Pasożyt **'wprowadź'** czyta program z urządzenia wejściowego. Jakiekolwiek komentarze nie grają żadnej roli w wykonaniu programu, są jedynie przechowywane do ewentualnego odtworzenia. Najogólniej składnię programów można opisać następująco:

```
<program>    : :=          ;
             : := <wyrażenie> <program>
<wyrażenie>  : := <komentarz>
             : := <reguła>
<komentarz>  : := <łańcuch>
```

Istnieje ważne ograniczenie w składni reguł programu. Służy ono do optymalizacji przeszukiwania zbioru reguł, pozwalając na bezpośredni do nich dostęp na podstawie identyfikatorów:

— każdy term, który stanowi lewą część reguły, musi zawierać co najmniej jedno wystąpienie identyfikatora, nie poprzedzonego zmienną, stałą lub < >
— żaden term, który stanowi lewą część reguły, nie może być postaci **t.s.** Wyjątek stanowi reguła predefiniowana **p.q→pq ;**

Dla wygody w manipulowaniu dużymi zbiorami reguł, dzieli się je na podzbiory zwane **światami.** Każdy z nich nosi nazwę składającą się z ciągu znaków:

M = c1, ... , cn

Świat o nazwie **M2** jest podświatem świata **M1,** jeśli **M1** i **M2** są postaci:

M1=c1 ,... , cm i M2=c1 ,... , cm , ... , cn

gdzie **n** jest większe od **m**. Jeśli **n=m+1,** mówimy o **podświecie bezpośrednim.** Taka notacja tworzy w sposób naturalny hierarchię światów, na szczycie której znajduje się świat, którego nazwa jest ciągiem pustym.

Z każdym wystąpieniem identyfikatorów związany jest pewien świat. To powiązanie tworzone jest w momencie wczytywania identyfikatorów, zgodnie z następującymi regułami:

— jeśli aktualny świat **M** jest podświatem świata **N** związanym z innym wystąpieniem tego samego identyfikatora, to związuje się **N** z wystąpieniem nowo wczytanego identyfikatora
— w przeciwnym przypadku, z wystąpieniem nowo wczytanego identyfikatora związuje się świat **M.**

Dzięki temu dwa wystąpienia identyfikatora złożonego z tego samego ciągu znaków nie są traktowane jako jednakowe, o ile nie jest z nimi związany ten sam świat. Dokładniej — wartością wystąpienia identyfikatora **id,** z którym związano świat **M, jest para (M,id).**

* * *

Kończąc tę pobieżną prezentację PROLOGU, chcielibyśmy zwrócić uwagę czytelników na fakt, iż naszym głównym celem było przedstawienie aktualnej wersji tego języka bez nadmiernego wgłębiania się w jego istotę i związaną z nim metodologię. Toteż powyższy opis traktować należy raczej jako uzupełnienie do książek [1, 4], bądź jako rodzaj wstępu do nich.

Niestety, w tak krótkiej prezentacji nie sposób było zmieścić tak ważnych zagadnień, jak podstawy teoretyczne PROLOGU, gramatyki metamorficzne itd., itd. Czytelnikom zainteresowanym podobną tematyką szczególnie polecamy zapoznanie się z raportem [2], z którego obficie korzystaliśmy przy opracowaniu naszych artykułów.

LITERATURA

[1] Clocksin W. F., Mellish C. S.: Programming in PROLOG. Springer-Verlag, 1981
[2] Colmerauer A., Kanoui H., Van Caneghem M.: PROLOG, Bases Théoriques et Developments Actuels, TSI, vol. 2, no. 4, p. 271—311, 1983
[3] Kluźniak F.: Remarks on Coroutines in PROLOG. w: IIUW Report 104, p. 19—30, 1981
[4] Kluźniak F., Szpakowicz S.: PROLOG. WNT, 1983.

Z tzw. przypadkami losowymi mamy do czynienia na co dzień. Nie chodzi tu bynajmniej o decyzje osobistego szefa, bo w takiej sytuacji mamy do czynienia z rozkładem nie tyle normalnym, ile tendencyjnym. Nie radzimy też nikomu próbować symulacji ruchu cen za pomocą niżej opisanych generatorów. Taki rozkład nie daje się po prostu opisać.

Ale kiedy realizujemy — na przykład — program do nauki pisania na maszynie, litery, które komputer każe nam naciskać, powinny być dobierane losowo. A przynajmniej takie wrażenie powinien mieć uczeń.

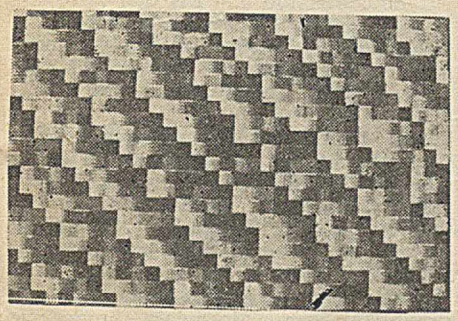
Programy edukacyjne i gry to najszerszy chyba zakres zastosowań generatorów liczb pseudolosowych. Nie należy jednak pomijać potencjalnych zastosowań w programach symulujących procesy lub zjawiska, wykorzystywanych do prognozowania. Niestety, na razie nic nam nie wiadomo o istnieniu takich programów na ZX SPECTRUM (może ktoś z Czytelników...).

Przedstawione poniżej generatory cechuje równomierny (z założenia) rozkład w przedziale (0,1). Dla zilustrowania tekstu wykorzystano znakomity pomysł firmy SINCLAIR umożliwiający ocenę „gołym okiem" jakości generatora. Kolejno generowane liczby przetwarzane są na kolorowe kratki na ekranie telewizora (na reprodukcjach są to niestety tylko odcienie szarości). Jeżeli na ekranie można dopatrzeć się jakiejś prawidłowości, to oznacza to, że generator jest „tendencyjny".

Generatory liczb pseudolosowych

ZX SPECTRUM posiada wbudowany generator liczb pseudolosowych wywoływany funkcją RND. Można go przetestować, zgodnie ze wskazówką w instrukcji obsługi, przy użyciu następującego programu:

```
10 POKE 22527+RND*704,RND*127
20 GOTO 10
```



1

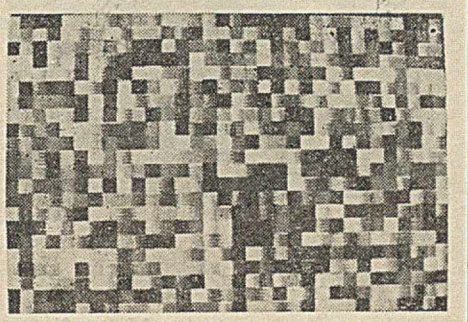
Zdjęcie 1 pokazuje wynik testu. Jak łatwo zauważyć jakość tego generatora jest słaba (widoczne skośne pasy!). W związku z tym napisano program umożliwiający przetestowanie różnych generatorów liczb pseudolosowych.

```
10 LET s=0.031019
20 LET r= FN q(s)
30 POKE 22527+s*704,r*127
40 LET s =FN q(r)
50 LET r=FN q(s)
60 GOTO 30
70 DEF FN q(x)= FN p(x) — INT
   (FN p(x))
80 DEF FN p(x) = 9821 *x+0.211327
```

Linie 70 i 80 definiują właściwy generator liczb pseudolosowych, tu:

$r_{n+1} = FRC(9821 * r_n + 0.211327)$

Generator ten jest znany jako jeden z najlepszych, a zarazem najprostszych [1].

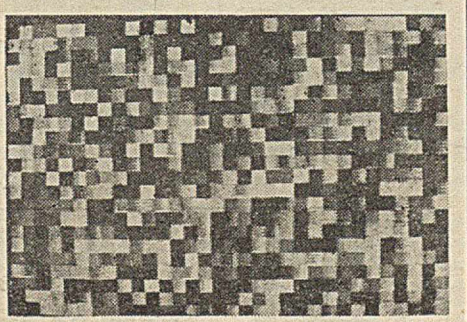


2

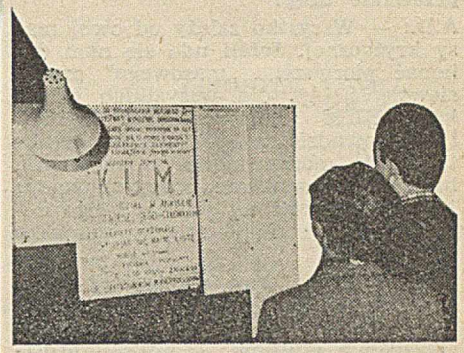
Zdjęcie 2 pokazuje, że rzeczywiście wygenerowana sekwencja wygląda „losowo". Proponujemy dla porównania przetestować i inne generatory:

```
80 DEF FN p(x)=(x+PI)**5
```

— wolny ale dobry — zdjęcie 3



3



Poprzez klub

Rozmowa z ANDRZEJEM DROŻNIAKIEM, Założycielem Klubu Użytkowników Mikroprocesorów

μK: — Polacy zajmujący się w Polsce mikroinformatyką — to na razie rzadkość. Raczkują też dopiero przyszłe ich organizacje. Czy dotychczasowe doświadczenie KUM-u pozwala na jakąś prognozę?

Andrzej Drożniak: — Perspektywa jest oczywista. Niezależnie od nazwy i form organizacyjnych, związki ludzi tej profesji i tych zainteresowań pojawią się na pewno. Nawet gdyby próbowano im przeszkadzać. Z drugiej zaś strony można ten rozwój przyspieszyć, pomagając sobie wzajemnie. Stąd pomysł KUM-u.

μK: — Oczywista jest perspektywa historyczna. Ale KUM jest propozycją na dziś...

A.D.: — KUM znalazł poparcie w Stowarzyszeniu Elektryków Polskich. A jest to na tyle potężna organizacja, że można liczyć na spory zastrzyk z tej strony. Zajmuje się nami Sekcja Maszyn i Systemów Cyfrowych.

Potrzebne jest teraz zebranie się pewnej masy krytycznej, podobnie jak w przyrodzie. W większości zjawisk musi zaistnieć pewna masa krytyczna i wtedy dopiero zaczyna się zasadnicza reakcja.

μK: — Poza KUM-em ten proces już chyba zachodzi!?

A.D.: — Tak, Perski Jarmark, tablica na Wydziale Elektroniki, gdzie można zdobyć niektóre potrzebne informacje, a w efekcie — części mikrokomputera bądź programy; tak, ale ten rynek nie wystarcza. Ciągle robi się w Polsce rzeczy elementarne, zamiast — mając zapewnione podstawy — dokonywać prób na wyższych poziomach. Potrzebne są więc takie utarte drogi, po których myśl konstruktora mogłaby przebić się przez pierwsze, banalne przeszkody.

Poprzez klub

µK: — I myśli Pan, że możliwe jest przetarcie dróg?

A.D.: — Wszystko zależy od owej masy krytycznej. Jeżeli uda się nam dokonać pierwszego „masowego" przedsięwzięcia (np. dla wszystkich członków KUM może zakupić BASIC i rozprowadzić po przystępnej cenie), to będzie można mówić o pokonaniu bariery. Elementarne oprogramowanie jest już przecież na świecie łatwo dostępne, a my ciągle robimy wszystko od podstaw.

µK: — Szuka Pan efektywnej drogi przy współudziale SEP. Stowarzyszenia NOT nie są chyba instytucjami szczególnie rzutkimi. A mikroinformatyka jest dynamiczna...

A.D.: — I młoda, to prawda. Większość osób zajmujących się tą dziedziną to ludzie młodzi.

µK: — Może więc potrzebne są też nowe instytucje?

A.D.: — Na razie takich nie widzę. Przed rokiem zwróciłem się do Polskiego Towarzystwa Informatycznego z propozycją współpracy. Mimo wstępnej zgody i wyraźnych oznak porozumienia, na razie niewiele z tego wyszło. Właściwie nie wiem dlaczego.

Nawiązałem też kontakt z klubem ABAKUS. Okazało się jednak, że nasze koncepcje są różne. Leszek Wilk, szef ABAKUSA, stara się dać użytkownikom gotowy sprzęt — tak, by mogli zajmować się wyłącznie oprogramowaniem, ja natomiast uważam, że wykształcenie w Polakach umiejętności samodzielnego, twórczego działania w tej dziedzinie wymaga znajomości budowy komputera, wiedzy konstruktorskiej.

µK: — Czyli w efekcie — po ogłoszeniu powstania Klubu — został Pan sam?

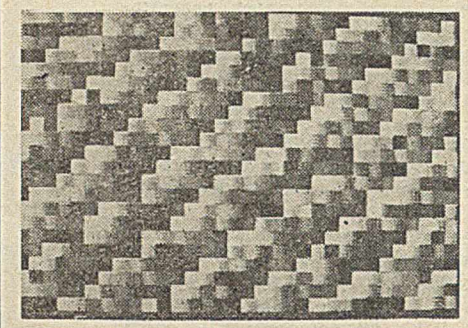
A.D.: — Nie, tak źle nie jest. Podstawową grupę tworzy kilkanaście osób. Ja sam gromadzę informacje o potencjalnych członkach, które niebawem — pocztą zwrotną — roześlę im wszystkim. Zgłosili się przede wszystkim amatorzy, jeszcze niewielu jest profesjonalistów i wytwórców. Mam nadzieję, że ta proporcja się zmieni. Staramy się zebrać dostateczną liczbę ofert od instytutów naukowych, pracowni rzemieślniczych, firm tworzących oprogramowanie, by w efekcie uzyskać możliwie kompletny obraz rynku.

µK: — Potrzebna będzie chyba jakaś forma organizacyjna.

A.D.: — Nie zdecydowaliśmy się jednak na założenie nowego stowarzyszenia. Główną formą działania będzie bowiem wymiana korespondencji. Jak dotąd rolę sekretariatu, czy może skrzynki kontaktowej, pełni mój dom i — po części — miejsce pracy. Potem działalność społeczna będzie zapewne marginalna. Dominować będzie — bo musi — interes każdego z nas. Wymiana oprogramowania, sprzedaż kostek czy płytek, diagnostyka... —

80 DEF FN p(x)=180/PI*x

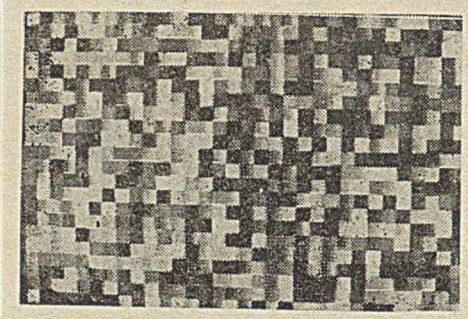
— bardzo słaby, ale łatwo programowalny na kalkulatorach, gdyż realizuje się za pomocą jednej funkcji R-D (zamiana radianów na stopnie) — zdjęcie 4



4

80 DEF FN p(x)=3579*x

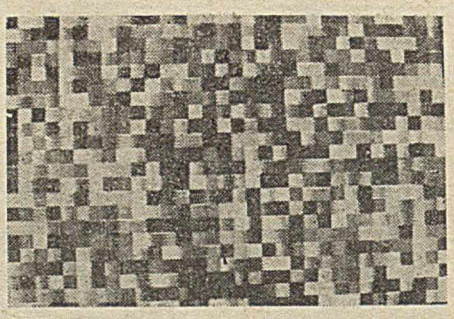
— oryginalny generator Marsagli, bardzo dobry i szybki, ale wrażliwy na wartość początkową — zdjęcie 5



5

80 DEF FN p(x)=997*x

— uproszczony generator Marsagli (po krótszym okresie następuje powtórzenie sekwencji, ale jest łatwiejszy w realizacji szczególnie dla programów napisanych w języku ASSEMBLER) — zdjęcie 6.

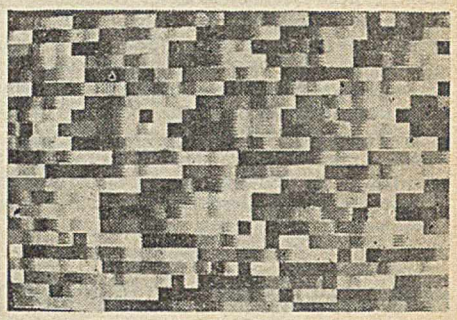


6

Wreszcie na końcu podam przykład generatora, który teoretycznie powinien dawać bardzo dobre rezultaty, ale w konkretnej realizacji (krótkie słowo maszyny, tzn. mała dokładność obliczeń) daje zupełnie nielosowy charakter wygenerowanego ciągu:

80 DEF FN p(x)=21*(x+PI)

Generator ten jest dobry, gdy dokładność obliczeń wynosi dziesięć cyfr [2] — zdjęcie 7.



7

Z przytoczonych przykładów wynika, że optymalny generator liczb pseudolosowych dla komputera ZX SPECTRUM ma postać:

```
10 LET s=0.031019
.
.   wywołanie w programie przez
    podstawienie s= FN q(s)
.
1000 DEF FN q(x)=FN p(x) —
     INT(FN p(x) )
1010 DEF FN p(x)=3579*x
```

Linia 1000 (oraz 70 w poprzednim programie) jest konieczna bo SINCLAIR BASIC nie ma instrukcji FRC (część ułamkowa liczby).

Tak więc stosunkowo łatwo można zastąpić maszynowy generator RND doskonalszym generatorem, lepiej przybliżającym losową sekwencję liczb. Ma to szczególne znaczenie we wszystkich pracach symulacyjnych, jak również w grach. Może nawet w tych ostatnich większe, gdyż nikt nie lubi przegrywać z powodu „oszustwa" maszyny!

JAKUB TATARKIEWICZ

LITERATURA

[1] Marsaglia G.: The structure of linear congruential sequences. In: Applications of Number Theory to Numerical Analysis, ed. S. K. Zabremba, Academic Press, 1972
[2] Moore R.: PPC Journal V6N2P20.

wyjściowe sterujące przełączanie. W systmie MSA-80.1000 spośród pozostałych bitów portu C, trzy wykorzystano do współpracy z magnetofonem kasetowym, a jeden do kasowania przerwania.

W systemie tym wykorzystano tylko 26 klawiszy (szesnaście dla wprowadzenia cyfr szesnastkowych oraz dziesięć jako klawisze funkcyjne). Oprócz nich w systemie zastosowano dwa klawisze nie wchodzące w skład matrycy: RESET — inicjujący system i INT — generujący przerwanie.

Oprogramowanie systemu polega na zorganizowaniu wysyłania danych na porty B i C oraz na wczytywaniu ich z portu A. Programowo realizowana jest też eliminacja efektu drgań styków klawiszy.

(cdn.)

ZBIGNIEW POJMAŃSKI
PIE, Warszawa

Wpisując program do pamięci komputera często nie uświadamiamy sobie, że oto właśnie ma miejsce k o m u n i k a c j a między nami a komputerem. Dzieje się to bezboleśnie, bowiem komunikację obsługuje program (Edytor, Interpreter, etc.), z którym współpracujemy (właśnie tak — wprowadzanie naszego programu do komputera możliwe jest dzięki realizacji innego programu, który jest już w pamięci komputera!). Aby umożliwić n a s z e m u programowi komunikację z użytkownikiem musimy zastosować specjalne instrukcje. Za ich pomocą program będzie mógł w odpowiednim momencie wprowadzić niezbędne informacje, jak również coś zakomunikować użytkownikowi (np. wynik realizacji programu). Takim właśnie specjalnym instrukcjom poświęcony został czwarty odcinek o języku FORTH.

FORTH (4)

Słowa używane do komunikacji program-operator:

KEY — pobranie znaku z klawiatury
KEY (— n)

Użycie słowa KEY powoduje wczytanie z klawiatury jednego znaku i zapisanie jego kodu (standardowo w kodzie ASCII) na stos danych.
Zobaczmy jaki kod ma litera A

KEY . [CR][A] 65 OK
Znak wczytywany z klawiatury słowem KEY n i e j e s t wyświetlany na ekranie.

EXPECT — pobranie ciągu znaków z klawiatury
EXPECT (n_2 n_1 —)

Użycie słowa EXPECT powoduje wczytanie z klawiatury ciągu znaków, zakończonego znakiem CR (jednak nie dłuższego niż wartość parametru n_1) i zapisanie kodów wczytanych znaków w pamięci począwszy od adresu wskazanego przez parametr n_2.
20000 10 EXPECT [CR] ABCD [CR] OK

W przypadku słowa EXPECT wczytywane znaki wyświetlane są na ekranie. Gdy wczytywany ciąg jest krótszy od n_1, zapis w pamięci uzupełniany jest zerami (jednak nie więcej niż trzy zera).

EMIT — wyświetlenie znaku
EMIT (n —)
Użycie słowa EMIT powoduje wyświetlenie na ekranie znaku, którego kod ASCII pobierany jest ze stosu.
65 EMIT [CR] A OK

TYPE — wyświetlenie ciągu znaków
TYPE (n_2 n_1 —)
Użycie słowa TYPE powoduje wyświetlenie n_1 znaków według kodów ASCII pobranych z pamięci, poczynając od adresu n_2.
20000 10 EXPECT 20000 10 TYPE [CR] ABDE [CR]
ABDE OK

Napotkanie przez słowo TYPE we wskazanym obszarze pamięci znaku końca napisu (trzy zera — jak przy zapisie przez słowo **EXPECT**) powoduje zakończenie wyprowadzenia znaków niezależnie od wartości n_1.

. " — wyświetlenie ciągu znaków

. " <napis>"
Użycie słowa . " powoduje wyświetlenie na ekranie ciągu znaków zawartych między ogranicznikami.
. " TEKST" [CR] TEKST OK
. " jest traktowane jak każde inne słowo i musi być oddzielane spacją, natomiast " jest traktowane jako ogranicznik i nie wymaga separacji.

FORMATY WYJŚCIOWE

FORTH przewiduje formaty wejściowe dla liczb podwójnej precyzji, bez znaku, tj. liczb 32-bitowych.

<# — otwarcie formatu
#> — zamknięcie formatu
#S — zapisanie liczby (lub pozostałej części liczby) jako ciągu cyfr
— zapisanie jednej cyfry
n **HOLD** — zapisanie (w środku liczby) znaku o kodzie n (ASCII).

Format opisuje zasady wyświetlania kolejnych cyfr liczby o d k o ń c a. Zamknięcie formatu powoduje odłożenie na stosie parametrów dla słowa TYPE, co umożliwia wyświetlenie liczby. Zapis /1000 oznacza nazwę słowa określającego format i polecenie wyświetlenia.
: /1000 <# # # # # 44 HOLD #S #> TYPE ;
12345678. /1000 CR 12345,678 OK
12. /1000 [CR] 0,012 OK

Liczby podwójnej precyzji zapisywane są z kropką. Zamiany innych liczb na liczby 32-bitowe można wykonać następująco:

: 31—>32 SWAP OVER DABS (moduł liczby podwójnej precyzji) ;
: 16—>32 0 ;
: 15—>32 DUP ABS (moduł) 0 ;

Po tym krótkim opisie kolejnych kilku słów, przedstawimy dwa proste przykłady.

Docierają do nas sygnały o wyrywaniu mikroKLANU z egzemplarzy INFORMATYKI znajdujących się w czytelniach. Nie jest to zbyt szczęśliwy sposób kontaktu z nami. Proponujemy jednak nadwerężyć kieszeń — rocznie 900 zł (bądź 600). Wszystkie zamówienia prenumeraty (p. str. 23) zostaną zrealizowane! 31 maja mija termin dokonywania wpłat na drugie półrocze.

PRZELICZANIE SEKUND NA GODZINY, MINUTY I SEKUNDY S—>GMS

```
: NN          (słowo pomocnicze)
  #           (zapisanie jednej cyfry)
  6 BASE !    (system szóstkowy jako podstawa we-wy)
  #           (zapisanie jednej cyfry)
  10 BASE !   (powrót do systemu dziesiętnego)
  58 HOLD     (zapisanie dwukropka)
;
: S—>GMS      (sekundy podawane jako liczba 32-bitowa)
  <#          (otwarcie formatu)
  NN          (zapisanie dwóch cyfr)
  NN          (zapisanie dwóch cyfr)
  #S          (zapisanie reszty liczby)
  #>          (zamknięcie formatu)
  TYPE        (wyświetlenie sformatowanej liczby)
;
```

Przykłady wykorzystania S—>GMS:

1000. S—>GMS [CR] 0:16:40 OK

60. S—>GMS [CR] 0:01:00 OK

15450. S—>GMS [CR] 4:17:30 OK

WIECZNY KALENDARZ DT

Program podaje dzień tygodnia dla wybranej daty.

```
: DT                          (na stosie dzień, miesiąc, rok)
  ROT OVER +                  (dni + lata)
  ROT DUP1 — 31 *             (m-1*31)
  >R ROT ROT R> +             (dni + lata + m + 31)
  OVER 100 / 1 + 3 4 * / —    (odliczenie stuleci)
  ROT ROT SWAP DUP 3 <        (styczeń lub luty?)
IF DROP 1 — 4 /               (odliczenie lat przestępnych dla stycznia/lutego)
  ELSE 4 * 23 + 10 /          (odliczenie miesięcy krótszych niż 31 dni)
  SWAP 4 /                    (lata przestępne)
  SWAP — THEN                 (razem)
  +                           (suma ogółem)
  7 MOD                       (modulo siedem)
                              (wyświetlenie nazwy dnia)
  DUP 0 = IF . " SOBOTA" ELSE
  DUP 1 = IF . " NIEDZIELA" ELSE
  DUP 2 = IF . " PONIEDZIAŁEK" ELSE
  DUP 3 = IF . " WTOREK" ELSE
  DUP 4 = IF . " ŚRODA" ELSE
  DUP 5 = IF . " CZWARTEK" ELSE
  DUP 6 = IF . " PIĄTEK" ELSE
  . " BLAD"
  THEN THEN THEN THEN THEN THEN THEN
  DROP                        (zgub miesiąc)
  CR                          (wyślij znak nowej linii)
```

Przykłady wykorzystania:

14 7 1410 DT [CR] WTOREK OK

1 4 1999 DT [CR] CZWARTEK OK

1 1 2000 DT [CR] SOBOTA OK

Proponujemy Czytelnikom przy pomocy poznanych słów przetłumaczenie na znaki literowe następującego ciągu kodów liczbowych: 80 82 73 77 65 32 65 80 82 73 76 73 83 0 0 0

MAREK CZARZYŃSKI
ABAKUS Warszawa

CENY • CENY • CENY • CENY • CENY

Marzec 1984

Wielka Brytania (ceny w funtach bez podatku)

Układy: 2716-4; 2732-3,5; 2764-5,75; 27128-18; 4116-1,2; 4164-4,5; 4532-3,5; 6502 CPU-3,5; 6522 VIA-3; 6532 RIOT-5,7; 6545 CRTC-10; 6800-2,2; 6809-6,3; 6845-6,5; 6800-68; 8080A-3,5; 8085-2,5; 8088-18; 8035-6; Z80 CPU-2,85; Z80 CTC-2,5; Z80 DART-6.5; Z80 DMA-7; Z80 PIO-2,5; Z80 SIO-8,5

Komputery osobiste: DRAGON 32K — 155; 64K-195; BBC-B-347; APPLE IIe — 600; BASE 64A (kopia APPLE IIe) — 349; MTX 500 (32K) — 275; MTX 512 (64K) — 315; NEW BRAIN A — — 225; A0-239; COMMODORE 64 — 184; Spectr-Video 318 — 173; 328—239; ORIC.1 — 129; ORIC ATMOS 43K — 170; TI 99/4 — 100; SPECTRUM 16K — 87; 48K — 113; ZX81 — 34,8; SINCLAIR QL — 399

- W tajemniczy sposób z Perskiego Jarmarku zniknęły prawie wszystkie układy związane z techniką mikroprocesorową. Zasięgnęliśmy języka — przyczyny posuchy obnażają rezultaty karkołomnej polityki rodzimego fiskusa. Otóż firmy polonijne zostały obciążone specjalnymi zobowiązaniami (w dewizach) na rzecz państwa za obroty w dewizach. Trudno więc się dziwić, że zmniejszają one swoje dolarowe obroty, a przez to maleje „mikroinformatyczna" podaż. W efekcie na Perskim wzrósł popyt, czego jarmark na razie nie wytrzymał. Kto na tym stracił? Po pierwsze — hobbyści, którzy stracili źródło zaopatrzenia. Po drugie — państwo, bo głównymi odbiorcami sprzętu mikrokomputerowego są firmy państwowe (ogromny popyt będzie zaspokojony w znacznie miejszym zakresie). Kto zyskał? Być może kilku sprzedających na Perskim (zwyżka cen), lecz zapewne nie Urząd Podatkowy... Komu więc dziękować za tę dalekowzroczną politykę obezwładniania mikroinformatyki?
- KUM. Udało się zrealizować rzecz na pozór niemożliwą. Do dyspozycji hobbystów jest tzw. system uruchomieniowy! Można więc za pomocą emulatora uruchomić samodzielnie skonstruowany mikrokomputer (niestety, pod warunkiem, że CPU to mikroprocesor 8080). W systemie uruchomieniowym został zainstalowany CP/M 1.4 — znakomite narzędzie przy uruchamianiu oprogramowania napisanego w języku ASSEMBLER 8080 (programy z 8030 nadają się również dla mikrokomputerów wykorzystujących Z80). W sprawie możliwości wykorzystania systemu należy się kontaktować z Andrzejem Drożniakiem — tel. dom. w Warszawie: 41-26-01. Uwaga: oferta kierowana jest do amatorów i korzystanie z systemu jest bezpłatne.
- Mikroinformatyka opanowuje dalsze połacie kraju. W Opolu powstała Pracownia Informatyczna „SPECTRUM" (zbieżność z ZX SPECTRUM zamierzona). Ambicją pracowni jest doprowadzenie do tego, aby każda szkoła w województwie opolskim została wyposażona w mikrokomputer (i oczywiście odpowiednie programy edukacyjne). Zanim rozwiną się podobne inicjatywy, w innych regionach, zachęcamy do wymiany doświadczeń: Pracownia Informatyczna „SPECTRUM", ul. Sienkiewicza 10/2, 45-037 OPOLE, tel. 345-51.
  Wprawdzie w Opolu nie powstał jeszcze klub mikrokomputerowy, ale w klubie MPiK realizowany jest przez PI „SPECTRUM" cykl spotkań, w czasie których każdy może spróbować swoich sił w szachach, brydżu, znajomości matematyki lub angielskiego — oczywiście za pomocą mikrokomputera.
- W klubie ABAKUS odbyło się... zebranie założycielskie. Członkowie klubu podjęli bowiem starania o rejestrację. Uzyskanie osobowości prawnej umożliwi zatrudnienie osoby, która będzie regularnie udostępniać sprzęt klubowiczom. W planach jest utworzenie pisma mikroinformatycznego.
- COMPUTER STUDIO KAJKOWSKI przygotowuje na krajowy rynek mikrokomputer kompatybilny z IBM-PC.
- Uwaga Miłośnicy FORTH'A! Firma Forth Interest Group oferuje ciekawe publikacje o tym języku (m.in. jak go zainstalować) płatne, niestety, w dolarach (rzędu kilkunastu). Czytelnicy mikroKLANU mogą otrzymać bezpłatnie odbitkę blankietu zamówienia (zawierającą spis i ceny), przysyłając w ciągu miesiąca od ukazania się tego numeru pod adresem redakcji:
  — zaadresowaną do siebie kopertę ze znaczkiem
  — odcięty róg okładki z napisem mikroKLAN 4
  — ewentualną ocenę publikacji o języku FORTH zamieszczonych w INFORMATYCE.
- Dla zmniejszenia deficytu finansowego INFORMATYKI zaproponowaliśmy, aby artykuły o sprzęcie krajowym sponsorowane były przez producentów. Dotychczas na nasz apel odpowiedziały wyłącznie firmy prywatne i to te, które nie mają najmniejszych problemów ze znalezieniem nabywców. Złośliwi twierdzą, że państwowi potentaci obawiają się wszelkiej wymiany informacji...
- Informacje mikroKLANU prosimy przesyłać — jak najszybciej — pod adresem INFORMATYKI: 00-041 Warszawa, ul. Jasna 14/16 p. 244, bądź telefonicznie: 27-71-40.

— prowadzi Andrzej J. Piotrowski (tel. dom. 48-22-85)

Prognozy rozwoju

Przy rozważaniu prognoz rozwoju należy rozróżnić procesy, które z dużym prawdopodobieństwem wystąpią, takie których wystąpienie jest możliwe oraz takie, które najprawdopodobniej się nie pojawią.

TENDENCJE BARDZO PRAWDOPODOBNE

Istnieje duże prawdopodobieństwo realizacji w najbliższych latach następujących zamierzeń, wynikających ze stanu zaawansowania prac badawczych.

Język naturalny

Od wielu lat autorka głosi zalety porozumiewania się ludzi z komputerem w języku naturalnym (tzn. języku ojczystym), zawierającym ewentualnie notacje naukowe. Jedną z największych zalet tej koncepcji jest łatwość, z jaką każdy mógłby przekazać komputerowi swoje żądania. Można wprawdzie twierdzić, że wraz z ogromnym rozpowszechnieniem nauczania informatyki, nie tylko w szkołach wyższych, lecz nawet w podstawowych, prawie każdy nauczy się programowania. Autorka uważa jednak, że prosty kurs języka BASIC (lub podobnego języka w przyszłości) nie może z każdego zrobić zawodowego programisty. Co więcej, nawet jeśli nie zabraknie doświadczonych programistów, to na pewno będziemy mieć coraz mniej czasu. Ten kto w tej sytuacji będzie mógł przekazać do komputera swoje żądania szybko i dokładnie (np. w języku naturalnym) zaoszczędzi więc wiele czasu i stanie się bardziej wydajny w swej codziennej pracy.

Jednym z najczęstszych błędów popełnianych przy rozważaniu problemu porozumiewania się z komputerem w języku naturalnym jest brak rozróżniania pomiędzy tym, co jest częściowym zapytaniem, a tym, co wymaga podjęcia określonego działania ujętego zwykle w postaci programu. Przykładem zapytania może być następujące zdanie:

PODAJ CZAS ODJAZDU WSZYSTKICH POCIĄGÓW Z PODUNK DO OSHKOSH, KTORE PRZYJEŻDŻAJĄ NA MIEJSCE PO GODZ. 18.00, ALE PRZED 20.00 I KTORE MAJA WAGONY RESTAURACYJNE.

Zapytanie tego typu może obsłużyć większość współczesnych systemów zarządzania bazami danych i nie wymaga skomplikowanej analizy językowej. Porównajmy to ze zdaniem zawartym w jednym z pierwszych podręczników FORTRANU, które w następujący sposób opisuje problem stanowiący podstawę do napisania programu:

DANY JEST PLIK KART. KAŻDA KARTA OPISUJE JEDNEGO MIESZKAŃCA PEWNEGO MIASTA. PRZECZYTAJ TE KARTY UŻYWAJĄC FORMATU (13,F10,2). LICZBA W PIERWSZYM POLU OZNACZA WIEK OSOBY, LICZBA W DRUGIM POLU TO JEJ DOCHÓD W 1960 R. NA OSTATNIEJ KARCIE PLIKU JEST —1 W PIERWSZYM POLU; TA INFORMACJA MA BYĆ WYKORZYSTANA DO SZUKANIA KOŃCA PLIKU KART. OBLICZ ŚREDNIĄ PŁACĘ OSÓB Z KAŻDEJ PIECIOLETNIEJ GRUPY WIEKOWEJ TZN: 0—4, 5—9, 10—14,..., 95—99.
WYDRUKUJ DOLNĄ GRANICĘ WIEKU W KAŻDEJ GRUPIE TZN: 0, 5, 10,..., 95, ŚREDNIĄ PŁACĘ DLA TEJ GRUPY I LICZBĘ OSÓB W TEJ GRUPIE. NALEŻY PODJĄĆ ŚRODKI ZAPOBIEGAJĄCE DZIELENIU PRZEZ 0 W PRZYPADKU GRUP PUSTYCH.

Przykład ten wymaga dokładnej analizy językowej opisu. Należy jednak podkreślić, że nawet używając języka naturalnego, trzeba podać wszystkie szczegóły i cechy zadania w sposób jasny i wyczerpujący. Dialog z komputerem może się okazać konieczny, tak jak konieczny bywa dialog między osobami, z których jedna nie rozumie problemu.

Języki do zastosowań specjalistycznych

Jeśli narzędzia pozwalające na porozumiewanie się z komputerem w języku naturalnym są niedostępne, to najlepszym rozwiązaniem jest komunikowanie się z maszyną w językach specjalistycznych. Są to wprawdzie języki sformalizowane, ale bardziej naturalne od takich języków programowania, jak np. PL/I. Najlepsze byłoby rozwiązanie, zapewniające każdej grupie użytkowników dysponowanie sztucznym językiem specjalnie dostosowanym do jej potrzeb. Terminologia bankowości jest przecież inna niż terminologia ubezpieczeń, a jedną z głównych cech specjalistycznego języka jest używanie zawodowego „żargonu" danej dziedziny. Powstanie w ciągu ostatnich lat znacznej liczby takich języków potwierdza, że jest to bardzo popularny sposób komunikacji z maszyną. We wszystkich prawie przypadkach twórcy tych języków musieli jednak wykorzystywać konwencjonalne metody syntaktyczne do definiowania języka, a implementacja była realizowana przez zbudowanie odpowiedniego kompilatora, interpretera lub — w kilku przypadkach — preprocesora.

Można mieć nadzieję, że stworzone zostaną narzędzia, dzięki którym użytkownik będzie określać tylko słownictwo i operacje odpowiednie dla danej dziedziny zastosowań oraz — wskazywać typ języka, jaki chciałby otrzymać (np. dopuszczający dowolny format, oparty na języku angielskim, o prostej notacji). Program maszynowy powinien następnie wygenerować definicję syntaktyczną języka i — po zaakceptowaniu jej przez użytkownika — automatycznie wyprodukować translator. Mimo że badania na ten temat trwają już od początku lat sześćdziesiątych, nie widać jeszcze szans jego bliższego rozwiązania.

Sposobem stworzenia pewnych udogodnień dla użytkownika wyspecjalizowanych systemów jest również tzw. metoda „menu". Polega ona zwykle na wyświetlaniu na ekranie listy możliwości, spośród których użytkownik wybiera następne działania. W pewnych przypadkach jest to rozwiązanie korzystne, choć ma te same wady, co wcześniej omówione pakiety programów użytkowych.

Weryfikacja programu

Ustalenie poprawności programu jest niewątpliwie jednym z najtrudniejszych problemów w dziedzinie programowania. Trudności sprawia nawet dokładna definicja „poprawności" — dla uproszczenia przyjmijmy, że poprawny program zawsze daje poprawną odpowiedź dla prawidłowych danych wejściowych i zawsze odrzuca nieprawidłowe dane wejściowe.

Poprawność programu należy rozważać na dwóch poziomach. Pierwszy — to rygorystyczne sprawdzenie pierwotnych intencji programu, tzn. ustalenie — przez porównywanie z początkowymi założeniami i wymaganiami — czy program rzeczywiście realizuje te założenia i wymagania. Drugi etap dowodu dotyczy samego programu oraz tego, co programista określił jako jego zadanie. Należy mieć nadzieję, że realizacja tego będzie miała charakter praktyczny, a nie tylko charakter interesującego ćwiczenia intelektualnego. Należy więc dążyć do opracowania metod użytkowych, tzn. takich, które można wykorzystywać do weryfikacji nie tylko małych i prostych programów.

W bardzo interesującym artykule z 1979 roku De Millo i współautorzy stwierdzili, że weryfikacja programu nigdy nie stanie się czynnością praktyczną, gdyż niemożliwe jest przeprowadzenie dowodu poprawności dużego programu w rozsądnie krótkim czasie. Autorzy uważają, że stworzeniu dowodu poprawności aktualnej wersji programu przeciwdziała sama natura cyklu programowania (obejmującego zarówno zmiany założeń, jak i samego programu), gdyż dowód nowej wersji wcale nie musi opierać się na dowodzie poprzedniej. Pogląd ten jest przygnębiający, choć prawdopodobnie słuszny.

Języki bardziej nieproceduralne

Nieproceduralność polega na tym, że użytkownik określa przede wszystkim c o ma być wykonane i nie podaje w j a k i s p o s ó b to osiągnąć. Działanie użytkownika po-

lega więc tylko na podaniu wymagań, na podstawie których odpowiedni system powinien najpierw wygenerować program rozwiązujący zadany problem, a następnie podać odpowiedzi na postawione pytania. Wiążą się z tym problemy wchodzące w zakres automatycznego programowania, języków do opisu specyfikacji, języków do definiowania problemów, języków bardzo wysokiego poziomu oraz języków bardziej nieproceduralnych. Ponieważ termin „języki nieproceduralne" z biegiem czasu zmienia swoje znaczenie, uda się być może osiągnąć niewielki postęp w tej dziedzinie. Innymi słowy — rozwiązanie tego szczególnego problemu podlega ewolucji i nie można go podsumować krótkim TAK lub NIE. Zwiększenie stopnia nieproceduralności jest jednak konieczne.

Bardziej teoretyczne podejścia

Programowanie, a w szczególności języki programowania, należałoby rozważać w sposób bardziej abstrakcyjny i teoretyczny. Chociaż od lat prowadzi się takie prace, to niestety pozostaje jeszcze bardzo wiele do zrobienia. Autorka sądzi, że programowanie powinno stać się mniej sztuką czy dyscypliną inżynierską, a bardziej nauką. Panują opinie, że programowanie jest — i powinno być — działalnością inżynierską, a odpowiednie wykorzystanie badań teoretycznych podniesie tylko jej jakość. Według autorki jednak programowanie (a ogólnie mówiąc — rozwój oprogramowania) jest nauką, która jeszcze nie zdołała się wykrystalizować.

Potrzeba dalszego rozwoju teoretycznego tej dziedziny jest oczywista, niezależnie od tego, czy na programowanie patrzy się jak na naukę, czy jak na dyscyplinę inżynierską. Oczekiwanie na rozwój programowania w językach naturalnych nie jest z tym sprzeczne, gdyż do jego rozwiązania potrzebna jest teoria, na której można by oprzeć metodę akceptowania i odpowiedniego przetwarzania takiej konwersacji z komputerem.

Wpływ na systemy wyszukiwawcze baz danych

Z każdym ze współczesnych systemów zarządzania bazą danych jest związany pewien „język", choć nie musi on być nazywany językiem i pozornie może go nie przypominać. Tym niemniej ludzie muszą porozumiewać się z bazą danych i sposób w jaki to robią wymaga naszej szczególnej uwagi. Ogólnie mówiąc — potrzebujemy języków zapytań ściślej definiowanych, bardziej elastycznych i bardziej naturalnych dla użytkownika.

Stosowanie języków wysokiego poziomu w systemach czasu rzeczywistego

Stare powiedzenie „szewc bez butów chodzi" na pewno odnosi się także do programistów systemowych, którzy większość prac wykonują w asemblerach lub — w najlepszym przypadku — w makrojęzykach lub językach pośrednich. Powodem tego jest stawiany im wymóg efektywności w odniesieniu do programów działających w czasie rzeczywistym oraz programów dla wbudowanych systemów komputerowych (zastosowania wojskowe). Wymog efektywności odnosi się zarówno do czasu wykonywania programów, jak i zapotrzebowania pamięci. Ten drugi problem może być rozwiązany przez wprowadzenie lepszego sprzętu, natomiast pierwszy jest nadal przeszkodą szczególnie trudną do pokonania. Przykładowo — w systemie kontroli lotów konieczna jest pewność, że system ten zakończy niezbędne obliczenia w odpowiednim czasie. Potrzebne są więc odpowiednio efektywne dla tego typu zastosowań języki i kompilatory.

TENDENCJE MOŻLIWE

Autorka spodziewa się dalszego rozwoju tendencji zarówno opisanych w poprzednim rozdziale, jak i całkowicie nowych.

Dalsze wykorzystywanie istniejących języków

Popularne obecnie języki, takie jak FORTRAN, COBOL czy BASIC, będą nadal szeroko stosowane. Prawdopodobnie zwolennicy każdego z nich będą w dalszym ciągu rozszerzać ich możliwości, w wyniku czego będą one miały bardzo zbliżone właściwości funkcjonalne, przy zupełnie różnej składni. BASIC stanie się jeszcze bardziej rozpowszechniony, dzięki poparciu ze strony jego obecnych użytkowników, a zwłaszcza przez wykorzystywanie go do programowania komputerów osobistych. PASCAL zdobędzie nowych użytkowników, a w końcu może zastąpi BASIC dla mikrokomputerów i komputerów osobistych (autorka nie wie jednak, który z tych dwóch języków będzie szerzej stosowany). APL będzie nadal popularny, choć prawdopodobnie nie zdobędzie już zbyt wielu nowych użytkowników.

LISP (przetwarzanie list), SNOBOL (przetwarzanie łańcuchów) oraz REDUCE i MACSYMA (przetwarzanie formuł) będą nadal powszechnie wykorzystywane w swych specyficznych dziedzinach zastosowań.

Niekontrolowane tworzenie nowych języków

Nowe języki programowania będą nadal powstawały w sposób niekontrolowany. Podana w drugiej części „Przeglądu" statystyka będzie nadal aktualna, gdyż co roku pewne języki giną, a w ich miejsce powstają nowe. Oczywiście chcielibyśmy, by każdy nowy język wzbogacał naszą dotychczasową wiedzę. Niestety, na każdy liczący się język, taki jak np. PASCAL, przypada co najmniej 20 innych, powodem powstania których było zaspokojenie ambicji projektanta lub dostarczenie tematu pracy doktorskiej.

Autorka spodziewa się, że stopniowo będzie wzrastać liczba języków dla specjalnych zastosowań. Nie sądzi jednak, żeby było to zjawisko niepożądane pod warunkiem jednak, że będą to nowe języki dla nowych zastosowań.

Wzrost nieproceduralności

Autorka wierzy, że zwiększy się nieproceduralność języków, tzn. stworzone zostaną języki, w których użytkownik będzie mógł podawać mniej szczegółów. Największe zmiany nastąpią prawdopodobnie w dziedzinie baz danych, ale jest to tylko jedna z dziedzin zastosowań, choć w obecnej chwili najbardziej popularna. Jeśli nie wydarzy się nic szczególnego, to zmiany w innych dziedzinach zastosowań będą jednak nieznaczne.

Weryfikacja nie stanie się użyteczna

Weryfikacja programów, mimo intensywnie prowadzonych prac badawczych, nie stanie się nigdy — według autorki — czynnością o cechach użytkowych. To przykra konstatacja, bowiem weryfikacja programów mogłaby mieć równie duży wpływ na rozwój oprogramowania, jak tranzystory i obwody scalone na rozwój sprzętu. Sytuacja, w jakiej znajduje się weryfikacja programów, jest podobna do opisanej w 1969 roku przez Lucas'a i Walk'a sytuacji metody VDL Vienna Definition Language). Najwięksi specjaliści w dziedzinie języków programowania i kompilatorów stwierdzili, że VDL jest wspaniałą metodą pozwalającą na dokładne zdefiniowanie języka i zbudowanie spójnego kompilatora. Niestety, VDL okazał się niepraktyczny z ekonomicznego punktu widzenia do pisania kompilatorów, które muszą mieścić się w ramach z góry określonych planów i środków.

Zwiększone wykorzystywanie teorii

Autorka przewiduje większe wykorzystywanie teorii, a nawet uważa to za jeden z najistotniejszych elementów prognoz. Coraz więcej ludzi uzyskuje wykształcenie w dziedzinie informatyki, a więc coraz większy potencjał intelektualny może wykorzystywać jej metodologię. Dobre wykształcenie teoretyczne może pomóc np. przy prześledzeniu cyklu życia oprogramowania i dokładnym określeniu sposobu przepływu i przekształcania informacji począwszy od wstępnych założeń aż do fazy jego konserwacji. Znaczącym przykładem jest też automatyczne generowanie (na podstawie formalnych specyfikacji) efektywnych kompilatorów dla dużych języków — prace nad tym problemem trwają od lat sześćdziesiątych, bez uzyskania ostatecznych rezultatów (nie rozwiązano nawet prostych przypadków). Wreszcie implikacje ogólności danych zaczynają być dopiero dostrzegane, co może doprowadzić do trudnych do przewidzenia skutków.

Lepsze języki zapytań

Autorka sądzi, że eksperci od języków wysokiego poziomu będą mieli duży wpływ na rozwój języków zapytań, ale na znaczące rezultaty w tej dziedzinie trzeba będzie jeszcze dość długo czekać.

Szersze wykorzystywanie języków wysokiego poziomu

Autorka uważa, że języki wysokiego poziomu będą używane znacznie szerzej, nawet w tak krytycznych zastosowaniach, jak programowanie w czasie rzeczywistym. Zwiększające się koszty cyklu życia skomplikowanych rozwiązań, takich jak systemy operacyjne, obronne czy sterowania rafinerią, spowodują, że prawie całość oprogramowania będzie napisana w językach wysokiego poziomu. W wielu przypadkach jedynie części krytyczne wymagające zaoszczędzenia każdej nanosekundy, będą kodowane w asemblerze.

TENDENCJE MAŁO PRAWDOPODOBNE

Ostatnią grupę przewidywań stanowią tendencje, których realizacja jest wg autorki mało prawdopodobna.

Odejście od języków rozszerzalnych

Języki rozszerzalne (ang. extensible language) przestały być głównym tematem prac badawczych. Autorka nie sądzi, żeby podjęto go na nowo, choć w niektórych pracach nad językami zwraca się uwagę na te metody, a w projektowanych językach wprowadza się elementy dopuszczające prostą rozszerzalność. Np. w ALGOLU 68 można definiować nowe operatory a w ADZIE istniejącym operatorom można nadawać nowe znaczenia. Autorka nie zgadza się natomiast z poglądem, że ogólność danych (a nawet podprogram) jest formą rozszerzalności. Za rozszerzalne uważa ona języki, do których — opierając się na istniejących konstrukcjach — można dodać nowe elementy syntaktyki (i związanej z nią semantyki).

Autorka ubolewa nad brakiem rozwoju w tej dziedzinie, gdyż języki rozszerzalne mogłyby być narzędziem pozwalającym na wygenerowanie nowych języków do specjalnych zastosowań. Niestety, wyniki na tym polu — w porównaniu z oczekiwaniami z przełomu lat sześćdziesiątych i siedemdziesiątych — są tak zniechęcające, że może to trwale, a co najmniej na długi czas przerwać prowadzone prace badawcze.

Brak specjalnych języków dla systemów rozłożonych

Autorka nie sądzi, ażeby mogły powstać specjalne języki dla systemów rozłożonych czy sieci informatycznych. Być może systemy operacyjne rozszerzy się o pewne elementy, ale większość problemów dotyczących języków wykorzystywanych w systemach rozłożonych odnosi się również do języków wykorzystywanych poza sieciami.

Brak nowych odkrywczych pomysłów

Autorka nie oczekuje żadnych praktycznych zmian wynikających z odkryć teoretycznych. Bez względu na zalety nowych koncepcji, jak np. programowanie funkcjonalne Backus'a (1978), tworzenie oprogramowania jest tak opracowane pod względem technologicznym, że aby zmienić dotychczasowy sposób postępowania trzeba by czegoś więcej niż jednego odkrywczego pomysłu. Inaczej mówiąc — autorka uważa, że przyszłe zmiany nastąpią poprzez ewolucję techniczną, a nie rewolucję teoretyczną.

ADA nie zastąpi innych języków

Autorka nie przypuszcza, żeby ADA wyłączając obszar działania Departamentu Obrony USA, zastąpiła inne języki. Prawdopodobnie język ten będzie szeroko stosowany, lecz inne języki mają zbyt wielu użytkowników, by można było je całkowicie wyeliminować.

* * *

Przegląd ten, rozpoczęty definicjami i krótką historią ostatnich dwudziestu lat, zawarł opis różnych problemów dotyczących języków wysokiego poziomu: środowiska, w których prowadzi się prace badawcze, rozwój inżynierii i metodologii oprogramowania, krótkie charakterystyki pięciu „nowoczesnych" języków oraz tło, na jakim powstawał najnowszy (choć nie najnowocześniejszy) z nich — ADA, tematy obecnych prac badawczych i wreszcie — przyszłe kierunki rozwoju.

Patrząc na historię i na przyszłość języków wysokiego poziomu, autorka dochodzi do jednego, niezbyt odkrywczego wniosku, że jedynym sposobem wykorzystania coraz większego potencjału sprzętu jest znalezienie łatwiejszych metod porozumiewania się z komputerem. W przyszłych zastosowaniach bowiem mniejszym problemem będzie efektywność sprzętu. Rozstrzygające znaczenie będzie miała komunikacja z komputerem, a prace nad językami wysokiego poziomu pozostaną jednym z zasadniczych tematów działalności badawczej w informatyce.

Opracowały:
HALINA CIECHOMSKA, TERESA WÓJCIEKIAN

na podstawie referatu Jean E. Sammet
przedstawionego na konferencji SEAS AM' 82

CSK–COMPUTER STUDIO KAJKOWSKI

Produkcja mikrokomputerów
Oprogramowanie
Doradztwo informatyczne

81-505 GDYNIA–Orłowo
ul. Balladyny 3b
(zakład)

81-651 GDYNIA
ul. Konwaliowa 10 m. 25
tel. 24-01-50

Nareszcie dostępne w kraju! Oprogramowanie użytkowe niezbędne w każdej nowoczesnej firmie. Łatwe w obsłudze nawet dla nieinformatyków.

BANK DANYCH — CSK — system zarządzania relacyjną bazą danych,
TABPLAN — CSK — pakiet do planowania, sporządzania kalkulacji, zestawień i sprawozdań,
TEKST — CSK (w opracowaniu) — system redagowania i edycji tekstów oraz programów źródłowych,
TRANSCOM — CSK — system współpracy mikrokomputera z komputerami ODRA, MERA, RIAD.

CSK dostarcza gotowe pakiety programowe, umożliwiające tworzenie użytkowych systemów przetwarzania danych w dowolnym obszarze zarządzania — dla mikrokomputerów typu ELWRO 513, IMP-85, ROBOTRON 5120, LIDIA lub IBM PC, pracujących pod nadzorem systemów operacyjnych CP/M 2.2. lub CP/M 86.

Cechą charakterystyczną tych systemów jest łatwość obsługi. Specjaliści różnych dziedzin, nie przygotowani zawodowo do pracy z komputerem — mogą tworzyć systemy informatyczne.

Przykładowe systemy użytkowe, które można zbudować przy wykorzystaniu wymienionych produktów programowych, to:

- **systemy płacowe, finansowo-księgowe, magazynowe, osobowe, bibliograficzne** — zakładane, uruchamiane i eksploatowane za pomocą BANK DANYCH — CSK,
- **systemy planowania, kalkulacji, szybkiego sporządzania ofert, sprawozdań, zestawień** — zakładane i eksploatowane za pomocą TABPLAN — CSK,
- **systemy automatycznego prowadzenia korespondencji, pisania dowolnych tekstów i ich edycji, redagowania programów źródłowych** — tworzone za pomocą TEKST — CSK,
- **systemy współpracy z komputerami ODRA 1305, MERA 400 jako teletype lub stacja ICL 7020 za pośrednictwem sieci telefonicznej lub specjalizowanych sieci telekomunikacyjnych** — tworzone za pomocą TRANSCOM — CSK.

CSK dostarcza wyżej wymienione systemy bazowe z pełną dokumentacją użytkową w cenach od 100 tys. do 400 tys. zł — w zależności od obszaru zastosowań, konfiguracji mikrokomputera i dodatkowych wymagań użytkownika. Możliwe jest także opracowanie i wdrożenie konkretnego systemu użytkowego na zlecenie klienta.

BANK DANYCH – CSK

Jest to system zarządzania relacyjną bazą danych, pomyślany jako narzędzie dla tworzenia systemów użytkowych. Został opracowany przez CSK dla szerokiej rodziny mikrokomputerów wykorzystujących mikroprocesory 8080 i Z80.

Wielkość zbiorów danych przetwarzanych w systemie bazy danych ograniczona jest następującymi parametrami:
— liczba rekordów w zbiorze — maks. 65 535,
— liczba znaków w rekordzie — maks. 1000,
— liczba znaków w polu — maks. 254,
— liczba pól w rekordzie — maks. 32,
— liczba znaków klucza indeksowego — maks. 100.

Dane mogą mieć charakter numeryczny i tekstowy.

Wymagania sprzętowe:
— mikrokomputer oparty na mikroprocesorze 8080, 8085, Z80,
— system operacyjny CP/M lub kompatybilny,
— min. 48 KB pamięci operacyjnej.

Wymagania te spełniają m.in. następujące mikrokomputery: LIDIA, ELWRO 513, IMP 85, ROBOTRON 5110 i 5120.

EO/557/K/84

TABPLAN – CSK

Jest to pakiet programowy oparty o koncepcję ELEKTRONICZNEGO FORMULARZA. Umożliwia pracę na arkuszu o wymiarach 63 kolumny i 255 wierszy. Posiadając możliwość definiowania pól tworzących dany formularz (3—32 znaki) jako opis (pole alfanumeryczne) — pozwala na łatwe tworzenie systemów informatycznych.

Użytkownik tworzy systemy bezpośrednio na swoim stanowisku pracy, wybierając jedną z wielu podanych na ekranie funkcji: edycja, usuwanie, wstawianie, kopiowanie, przenoszenie, sortowanie, formatowanie, skalowanie, definiowanie, zerowanie, drukowanie itp. Dotyczą one zarówno pojedynczych pól, wierszy, kolumn, jak i całego formularza. Zaprojektowane i wypełnione formularze (kalkulacyjne, planistyczne, sprawozdawcze itp.) można zapisać na dysku lub wydrukować na drukarce.

Wymagania sprzętowe analogiczne jak dla BANKU DANYCH — CSK.

TRANSCOM–CSK (EMULATOR TTY)

Emulator TTY umożliwia współpracę mikrokomputera z komputerami serii ODRA 1300 w trybie MOP, pod kontrolą systemu operacyjnego GEORGE-3. Mikrokomputer, pracujący pod kontrolą systemu operacyjnego CP/M, zastępuje w tym układzie urządzenie typu dalekopis, a ponadto umożliwia transmisję zbiorów:
— typu tekst (w systemie CP/M) do zbiorów typu GRAPHIC (w systemie GEORGE-3),
— typu GRAPHIC (w systemie GEORGE-3) do zbiorów typu tekst (w systemie CP/M).

Emulator TTY umożliwia użytkownikowi dostęp do dużej maszyny cyfrowej w celu wykonania na niej obliczeń wymagających zaangażowania znacznych zasobów czasu procesora, pamięci operacyjnej i pamięci zewnętrznych. Po wykonaniu przetwarzania na komputerze, wynikowy zbiór może być przesłany z powrotem do mikrokomputera.

Wymagania sprzętowe analogiczne jak dla BANKU DANYCH — CSK.

CSK specjalizując się zarówno w sprzętowych, jak i programowych problemach grafiki mikrokomputerowej — oferuje:

USŁUGI SPRZĘTOWE:

- pomoc przy wyborze i zakupie ploterów oraz urządzeń do digitalizacji,
- przyłączanie tego typu urządzeń do komputera (w tym — wszystkich krajowej produkcji),
- serwis i naprawy gwarancyjne ploterów firm:

WATANABE (Japonia, RFN)
GOERZ (Austria),

- przystosowywanie telewizorów JOWISZ i NEPTUN do funkcji monitorów kolorowych z wejściem RGB.

OPROGRAMOWANIE NARZĘDZIOWE:

- do rozszerzania dowolnego języka programowania o instrukcje graficzne i obsługi ploterów,
- do obsługi procesu automatycznej digitalizacji dokumentów, zdjęć, rysunków, szkiców itd.,
- do wspomagania procesu tworzenia grafiki „trójwymiarowej".

SYSTEMY UŻYTKOWE:

- wspomagania procesu projektowania płytek obwodów drukowanych,
- wspomagania procesu kreślenia rysunków technicznych maszynowych,
- redagowania tekstu TEKST CSK z ploterem, jako urządzeniem drukującym (edycja pisma o dowolnym kształcie i kolorze czcionki).

Oferowane oprogramowanie pracuje pod kontrolą systemów operacyjnych CP/M 2.2. oraz CP/M-86.

CSK zaprasza na Targi

56 Międzynarodowe Targi Poznańskie

10—17 czerwca 1984

pawilon 2, na piętrze

EO/557/K/34

R-35 w praktyce

INFORMATYKA opublikowała niedawno (nr 11/83, s. 36) notatkę — reklamówkę zatytułowaną „R-35 nowość Jednolitego Systemu". Ponieważ komputery R-35 pracują w Polsce od kilku lat, warto do tematu powrócić w celu zestawienia informacji reklamowych z sytuacją rzeczywistą.

Należy podkreślić, że R-35 supernowością nie jest (nowszy jest np. R-60); znany mi egzemplarz R-35 pracuje od roku 1980. Eksploatacja tego komputera nie nastraja tak optymistycznie jak mogłoby wynikać z przytoczonej notatki.

Prawdą jest, że każdy sprzęt prototypowy — a takim jest wspomniany egzemplarz R-35 — charakteryzuje stosunkowo duża awaryjność. Nie ma więc sensu ukrywać, że R-35 psuje się często, a czasem — bardzo często. O ile jednak pominiemy niedomagania „wieku niemowlęcego", pozostanie jeszcze kilka spraw, na które należy zwrócić uwagę.

W skład konfiguracji podstawowej EC 1035—01 wchodzi m.in. pulpit operatora wyposażony w elektryczną maszynę do pisania CONSUL. Urządzenie to — jak wykazała praktyka — psuje się bardzo często, a komunikacja z systemem jest wolna i niewygodna w pracy operatorskiej. Lepszym rozwiązaniem jest łączność przez monitor ekranowy co zresztą — w znanym mi ośrodku — zostało zrealizowane we własnym zakresie i doskonale zdaje egzamin.

W notatce podano, że producent dostarcza jednostki dyskowe o pojemności 100 M bajtów. Należy więc sądzić, że w tym przypadku zmieniany jest rodzaj kanałów przesłań: dyski — jednostka centralna. W konfiguracji podstawowej przesłania odbywają się bowiem w trybie szeregowym, a więc siłą rzeczy są wolne i współpraca z dyskami 100 M bajtów jest praktycznie niemożliwa (bardzo nieefektywna). Ogólnie zresztą — przesłania systemowe w R-35 są wolne, szczególnie wielokrotnie powtarzane operacje przesyłania stron znacznie zwiększają łączny czas realizacji programów. Parametry szybkości procesora są porównywalne z szybkością procesora R-32.

Przytoczone w notatce granice związane z warunkami pracy wydają się mocno przesadzone (5—40°C, wilgotność 40—95%). W szczegółowej dokumentacji technicznej producent zaznacza, że jednostka centralna powinna pracować w temperaturze 18—20°C i przy wilgotności 55—60%. Ograniczenia te potwierdziła praktyka; prawidłowością jest, że mimo dobrego systemu klimatyzacji, latem maszyna psuje się o wiele częściej.

Informacja o bogatym oprogramowaniu technicznym i nowoczesnym oprogramowaniu systemowym przerasta rzeczywistość. Rzeczą oczywistą jest, że najlepszy nawet test nie wykryje każdego możliwego uszkodzenia. Szkoda jednak, że oprogramowanie techniczne nie daje możliwości odnotowania błędów niestabilnych (chwilowych). Gorzej jest z oprogramowaniem systemowym. Często okazuje się, że rzeczy mieszczące się w dokumentacji oprogramowania systemowego i odpowiadające normom systemu IBM 370/RIAD — po prostu nie „chodzą".

Istnieje obecnie około dziesięciu wersji oprogramowania systemowego. W chwili zakupu wspomnianego egzemplarza R-35 (przypominam — rok 1980) producent dostarczył wersję drugą. Po długich i nieskutecznych walkach z oprogramowaniem, wyreklamowano wersję siódmą, która nie jest bynajmniej rozszerzeniem możliwości systemu operacyjnego, jest natomiast lepsza, bowiem bardziej przystaje do dokumentacji — jakkolwiek i w tej wersji są błędy. Nawiasem mówiąc dopiero od wersji szóstej — siódmej przewidziano możliwość teleprzetwarzania (pracę w sieci).

System operacyjny R-35 nie jest — jak dotąd — systemem OS/VS, lecz systemem OS/SVS. System ten nie jest odpowiednikiem znanego z rodziny 370 systemu operacyjnego VS (systemu z pełną wirtualizacją pamięci). lecz przypomina bardziej system OS/MVT (wieloprogramowanie ze zmienną liczbą zadań). OS/SVS producent nazywa wieloprogramowym systemem o zmiennej liczbie zadań wymiennie wykorzystujących pamięć wirtualną. Nie wdając się w dalsze szczegóły, należy stwierdzić, że system OS/SVS jest mniej efektywny w stosunku do OS/VS.

Oprogramowanie podstawowe R-35 obejmuje kompilatory języka ASSEMBLER, RPG, ALGOL, COBOL, FORTRAN, PL/1. Istnieje opinia, że kompilator ALGOLU jest niedopracowany — na ogół więc się z niego nie korzysta. FORTRAN ma minimalną bibliotekę standardowych procedur matematycznych. Diagnostyka błędów w tym języku jest niekompletna i często niejednoznaczna. Niektóre operacje matematyczne (np. potęgowanie) zrealizowane są nieefektywnie. ASSEMBLER i PL/1 nie budzą natomiast zastrzeżeń. Należy jeszcze dodać, że przenoszenie oprogramowania IBM 370/RIAD wymaga sporego wkładu pracy.

*

A teraz kilka szerszych uwag.

Kłopoty związane z pracą urządzeń pomocniczych produkowanych przez kooperantów Jednolitego Systemu, wskazują że nie wszystko jest dopowiedziane do końca. Reklamówka urządzenia graficznego DIGIGRAF informuje — na przykład — że może ono pracować w trybie bezpośrednim z komputerem. Natomiast producent procesora (R-35) nie zgadza się, pod groźbą utraty gwarancji, na dokonanie takiego połączenia.

Biblioteka programowa DIGIGRAFU jest bardzo uboga. Na marginesie dodam, że nie wszystkie procedury graficzne zakupionej — w znanym mi ośrodku obliczeniowym — wersji oprogramowania były poprawne. Należałoby się poza tym zastanowić, dlaczego utworzono taką bibliotekę — gdy od kilku już lat użytkownicy urządzeń graficznych wymieniają pomiędzy sobą zupełnie dobre oprogramo-

wanie graficzne sprowadzone z Zachodu. Biblioteki te przystają ponadto do różnych pakietów specjalistycznych wyszperanych przez hobbystów-informatyków.

Jednolity System miał nas uniezależnić od zachodnich producentów sprzętu informatycznego. Dlaczego więc blok pamięci DIGIGRAFU jest produkowany wyłącznie w Stanach Zjednoczonych i w przypadku awarii nie da się bezpośrednio zastąpić częścią dostępną na naszym rynku?

Pozostaje jeszcze sprawa odpowiedniego serwisu oraz gospodarki częściami zamiennymi. Pod tym względem również nie jest dobrze. Spowodowane jest to zapewne trudnościami związanymi z ogólną sytuacją gospodarczą kraju. Wiele niedogodności wynika jednak — jak wskazuje praktyka — z lekceważenia normalnych obowiązków i bałaganiarstwa. Przydałby się ktoś kontrolujący prawidłowość zawarcia oraz proces realizacji kontraktów przy konkretnych zakupach sprzętu informatycznego, bowiem patrząc od dołu ma się wrażenie zupełnego chaosu. Przykładowo — czas dostawy części zamiennych w trybie awaryjnym ustalono, w pewnym kontrakcie, na 25 dni (!), a i ten termin jest notorycznie niedotrzymywany.

Zadaniem przedstawicielstwa handlu zagranicznego zajmującego się sprzętem informatycznym jest pośredniczenie w kontaktach między producentami a użytkownikami i dbanie o interesy tych ostatnich. Zdarzało się, że pisma interwencyjne do producentów ginęły u pośrednika, czyli w przedsiębiorstwie METRONEX, co w efekcie przesuwało realizację napraw gwarancyjnych o miesiące. Już co najmniej od roku METRONEX obiecuje stworzenie ekipy serwisowej DIGIGRAFU...

*

Pomijając zagadnienie „rzeczywistości" R-35 — wydaje się, że powyższe fakty nie są wyjątkami. Mimo że przykładem był dla mnie jeden ośrodek obliczeniowy i jeden egzemplarz komputera Jednolitego Systemu, podejrzewam, a co więcej — kontakty z innymi użytkownikami systemu RIAD dają mi pewność, że przytoczone przykłady nie są wynikiem szczególnego „informatycznego" pecha.

Nie można również ukrywać, że system RIAD jest dla niektórych zastosowań przestarzały i bardzo niewygodny w eksploatacji. W zamieszczonym w INFORMATYCE (nr 10/83, s. 35) sprawozdaniu ze spotkania CERN, przytoczono taką oto uwagę uczestnika: „Jak fizycy mogą pisać dobre programy według prawidłowych wzorców, jeśli na co dzień mają do czynienia z najgorszym oprogramowaniem jakie kiedykolwiek stworzono z systemem operacyjnym IBM?" Wypowiedź ta daje do myślenia — zwłaszcza nam, niekoniecznie fizykom, którzy na co dzień borykamy się z komputerami Jednolitego Systemu, w zamierzeniach opartymi przecież na systemie IBM...

TERESA WILCZEK

TARGI BRNO '83

Na XXV Targach Brneńskich (wrzesień 1983) królowały mini- i mikrokomputery oraz urządzenia peryferyjne (automaty obrachunkowe, zestawy do obróbki tekstów, emisji dokumentacji itp.). Dominowały ekspozycje krajów RWPG, a zwłaszcza — czechosłowacka i węgierska.

Ekspozycja CSRS

Czechosłowacki przemysł przyjął na lata osiemdziesiąte szeroko zakrojony program badawczo-konstrukcyjny, produkcyjny i aplikacyjny mini- i mikrokomputerów systemu SM. Przygotowywane zastosowania wiążą się głównie z systemami sterowania procesami produkcyjnymi (technologicznymi). 8- i 16-bitowe mikrokomputery SM 50/40 i SM 50/50, bardzo wydajny minikomputer SM 52/11 oraz moduły procesorowe SM 53/10 i 53/20, które mogą wymienione systemy łączyć w zestawy hierarchiczne — będą seryjnie produkowane jeszcze w tym roku — podobnie jak wszelkiego rodzaju czujniki, konwertery i moduły wykonawcze. Są to systemy tzw. II generacji SM (SMEP II).

Obecnie pracuje ok. 200 zestawów minikomputerów I generacji (SMEP I), w tym wiele zestawów bardzo udanego modelu SM 4-20 z pamięcią 64 lub 128 K słów 16-bitowych (czas dostępu $\leqslant$ 600 ns, cykl pamięci $\leqslant$ 640 ns). Zestaw obejmuje następujące podstawowe urządzenia peryferyjne:

— monitory ekranowe lokalne i zdalne
— drukarkę znakową CONSUL (DZM 165) o szybkości ok. 40 wierszy/min
— 2—4 jednostki dysków kasetowych wymiennych o pojemności 5 MB
— do 4 jednostek pamięci taśmowej o szerokości 36 K znaków/s
— 2 dyskietki o pojemności 512 KB
— czytnik/dziurkarkę taśmy papierowej
— czytnik kart dziurkowanych.

Minikomputery SM 4-20 są instalowane w normalnych warunkach biurowych i — zdaniem użytkowników — mają niezawodną jednostkę centralną, natomiast niezbyt pewne w eksploatacji jednostki dyskowe. Stosowane są do obliczeń naukowo-technicznych oraz do przygotowywania i testowania oprogramowania systemów sterowania procesami technologicznymi. Mogą również tworzyć sieć terminali inteligentnych dużych komputerów. System SM 4-20 jest kompatybilny „w górę" z poprzednio stosowanym SM 3-20 (ma rozszerzony repertuar instrukcji oraz większą pamięć operacyjną).

Poza wymienionymi „typowymi" urządzeniami peryferyjnymi przewidziano możliwość przyłączenia specjalnych jednostek sterujących, np. dla systemu pomiarowego IMS-2, DASIO, grafoskopu, autokreślarki, czytnika współrzędnych (ang. digitizer), jednostek dyskowych 29 MB lub szybkiej drukarki wierszowej VIDEOTON (900 wierszy 136-znakowych na minutę).

SM 4-20 wyposażony jest w system operacyjny czasu rzeczywistego DOS RV2 oraz translatory języków MAKROASSEMBLER, FORTRAN IV (PLUS), COBOL, BASIC PLUS 2. System wyposażony jest ponadto w pakiety programowe sterowania sieciami komputerowymi i terminalowymi, terminalami graficznymi oraz do obliczeń naukowo-technicznych.

Minikomputer SM 52/11 jest od dawna zapowiadanym największym zestawem przeznaczonym do wielu zastosowań, jakkolwiek nie potwierdziły się na Targach wcześniejsze pogłoski o tym, że będzie on wyposażony w pamięć operacyjną o pojemności do 1 MB. Awizowany jest następny model o symbolu SM 52/11 PLUS, wyposażony w pamięć o pojemności do 4 BM, który ma być dostarczany do 1985 r. SM 52/11 jest konstrukcyjnie i programowo kompatybilny „w górę" z omówionymi SM 3-20 i SM 4-20.

Specjalnego podkreślenia wymaga fakt przygotowywania dla systemów SM 52/11 oraz SM 50/50-1 i SM 4-20 pakietu programowego MARKAB 33XL przeznaczonego do przygotowania, wstępnego przetwarzania oraz transmisji danych. Stanowiska operatorskie mogą być wyposażone w monitory ekranowe lub drukarkę mozaikową z klawiaturą. Systemy te mają zastąpić stosowane dotąd w CSRS zestawy SEECHECK, R850 lub PERTEC.

Poza wspomnianymi systemami produkowane są małe zestawy CONSUL 2711, 2712, 2713 i 2714 o zróżnicowanej konfiguracji do zdecentralizowanego przygotowania i wstępnego przetwarzania lub transmisji danych. Przeznaczone są do rejestracji danych przez 1—2 operatorów na 2 dyskach elastycznych (1898 bloków po 128 znaków każdy). Ekran umożliwia wyświetlenie 4 wierszy po 40 znaków. Wyposażone są w pamięć półprzewodnikową RAM o pojemności 2 KB oraz ROM o pojemności 6 KB. Niektóre mogą być wyposażone w drukarkę znakowo-mozaikową oraz w jednostkę wolnej pamięci taśmowej (10 KB/s).

Mogą stanowić samodzielny lub współpracujący z komputerem zestaw do prostego przetwarzania danych bezpośrednio na stanowisku pracy. Całym zestawem steruje system operacyjny zapewniający organizowanie procesu oraz programową kontrolę danych. Programy własne mogą być opracowywane w języku BAL.

Mikrokomputer SM 50/50, oparty na mikroprocesorze 16-bitowym, charakteryzuje się szybkością 400 tys. operacji na sekundę. Pamięć operacyjna do 256 KB, w tym typu „Cache" 2

KB oraz 16 K RAM i 12 K EPROM. Podstawowe urządzenia peryferyjne: dyski elastyczne oraz 8 monitorów ekranowych. Zestaw przeznaczony głównie do małych systemów sterowania produkcją oraz zbierania i wstępnego przetwarzania danych.

Jednak największym szlagierem wystawy był mikrokomputer SM 53/10. Kandydował do złotego medalu Targów i robił wrażenie nawet na laikach — dzięki zainstalowanemu kolorowemu monitorowi ekranowemu, umożliwiającemu bardziej atrakcyjną prezentację informacji graficznych. Przeznaczony jest do sterowania procesami technologicznymi w rozłożonych przestrzennie sieciach hierarchicznych. Dla tego zestawu przygotowywany jest uniwersalny pakiet programów o nazwie MODUS, przeznaczony do obsługi systemów sterowania lub zarządzania produkcją, a także informacyjnego wspomagania takich systemów.

Podstawowymi składnikami konfiguracji SM 53/10 są:

- terminal operatora procesu TOP
- terminal sterowania procesem TSP
- mikroprocesor komunikacyjny 50/40-1 ILPS
- magistrala systemu SM 53/10.

TOP zbudowany jest w oparciu o mikroprocesor SM 50/40-1 i przeznaczony do komunikacji operatora z systemem 53/10. Do wprowadzania danych i poleceń oraz do regulacji systemu służy klawiatura z oprogramowanymi funkcjami. Informacje wejściowe z systemu wyprowadzone są na czarno-białym lub kolorowym monitorze quasi-graficznym (32 wiersze po 64 znaków) lub na drukarce mozaikowej oraz na wyświetlaczu klawiatury. Dane normatywne (regulacyjne) zapisane są na dyskach elastycznych. TOP realizuje m.in. następujące funkcje: kontrola wybranych wielkości technologicznych oraz przebiegu procesu, sterowanie elementami nastawczymi, prezentacja odchyleń i awarii, rejestracja produkcji. TSP jest skonstruowany również w oparciu o mikroprocesor 50/40-1 i jest przeznaczony do bezpośredniego sterowania procesami poprzez czujnik oraz elementy regulacyjne. Jest on kontrolowany jedynie przez TOP lub system centralny i realizuje następujące funkcje:

- zbieranie danych z procesu
- wstępne przetwarzanie danych technologicznych
- wyprowadzanie do procesu zadanych lub wyliczonych wielkości
- regulacja i logika sterowania
- komunikacja z TOP i systemem nadrzędnym
- diagnostyka systemu.

Moduły pamięciowe TOP i TSP mogą być dostarczone w różnych wariantach, np. 64 KB RAM dynamiczne + 16 KB EPROM, 16 KB RAM dynamiczne, 16 KB EPROM, 16 KB CMOS statyczne.

Wszystkie terminale są połączone za pomocą wybudowanych procesorów komunikacyjnych ILPS oraz kabli o długości do 1,5 km.

Ponadto w ekspozycji CSRS zaprezentowano następujące nowe urządzenia:

- pamięć dyskową o pojemności 100 MB, którą można przyłączyć do wszystkich komputerów RIAD lub SM za pośrednictwem odpowiednich jednostek sterujących
- kalkulator biurowy SP830 z pamięcią 16—48 KB, monitorem ekranowym i drukarką wierszową oraz całą gamą innych urządzeń zewnętrznych (czytnik współrzędnych, pamięć taśmowa, kasetowa itp.).

Ekspozycja Węgier

Ekspozycja ta wzbudzała również zainteresowanie, mimo że główną jej pozycją był system EC 1011 [1]), który można uznać za mikroprocesorowego następcę poprzednio produkowanego na Węgrzech R-10. Parametry techniczne systemu są imponujące:
— pamięć operacyjna do 1 MB
— pamięć dyskowa 50 MB (produkcji rumuńskiej na licencji CDC)
— pamięć dyskowa stałogłowicowa 0,8—2,5 MB
— czytnik kart (600 kart/min)
— pamięć na dyskach elastycznych 255 lub 380 KB
— drukarki wierszowe: 300, 600, 900 lub 1200 wierszy/min
— drukarka mozaikowa 180 znaków/min
— pamięć taśmowa 800 lub 1600 bpi
— monitory ekranowe VIDEOTON.

System EC 1011 jest wyposażony w podstawowe języki programowania: MAS (makroassembler), FORTRAN IV, RTL, COBOL, BASIC, MAG (makrogenerator). Przewidywane zastosowanie to zarówno przetwarzanie danych oraz tworzenie sieci komputerowych, jak i sterowanie procesami technologicznymi.

Jak wiadomo, Węgry są od dawna znanym producentem urządzeń peryferyjnych: monitorów ekranowych, drukarek, mini- i mikrokomputerów do sterowania procesami technologicznymi oraz obliczeń naukowo-technicznych.

Ekspozycja Polski

Mimo skromnego wyposażenia i peryferyjnego usytuowania — prezentowała się interesująco, dzięki nowoczesnemu mikrokomputerowi z serii ELWRO 500. Jednopłytowa jednostka centralna o symbolu 01/A zawiera następujące układy:
— mikroprocesor 8-bitowy (odpowiednik INTEL 8080A)
— pamięć ROM/EPROM 6—12 KB (INTEL 2708)
— pamięć RAM 4—48 KB (INTEL 2114)
— 4 kanały równoległego we/wy oraz kanał transmisji szeregowej.

Urządzenia peryferyjne to:

— monitor ekranowy (16 wierszy × × 64 znaków) z klawiaturą
— drukarka ROBOTRON 1152 (specjalnie przeznaczona do automatów księgowych)
— przystawka do stosowania kart kontowych o szerokości 145—405 mm
— 2 jednostki pamięci na dyskach elastycznych 256 KB.

Oprogramowanie obejmuje:
— podstawowy system operacyjny z interpreterem języka ZIM
— dyskowy system operacyjny (kompatybilny z CP/M).

Producent przewiduje całą gamę modeli w serii 500 oraz szerokie zastosowanie w przedsiębiorstwie jako urządzenie autonomiczne (np. księgowość, płace, gospodarka materiałowa, fakturowanie) lub jako inteligentnego terminala dla komputerów serii RIAD lub SM.

Ekspozycja NRD

Obejmowała trzy pozycje: minikomputer A 5130, mikrokomputer K 1520 oraz terminal bankowy K 8924.

A 5130 stanowiący ewolucje modelu A 5120 może mieć pamięć o pojemności 64 KB, dwa typy drukarek mozaikowych: (30 znaków lub 100—400 znaków/s). Do drukarek stosowany może być papier „z wałka", składanka komputerowa lub odrębne karty kont księgowych. Przyłączone mogą być 2 typy monitorów ekranowych o pojemności 1024 lub 1920 znaków oraz pamięć na dyskach elastycznych (pojemność do 1 MB) lub pamięć kasetowa a także urządzenia transmisji danych. A 5130 wyposażony jest w system operacyjny SIOS oraz kompilatory języków COBOL i PASCAL. Określany jest jako automat biurowy do typowych zastosowań ekonomicznych lub jako inteligentny terminal.

Mikrokomputer K 1520 jest ewolucją modelu K 1510. Oparty jest na 8-bitowym mikroprocesorze i wyposażony w pamięć 4 KB MOS (konstrukcyjnie procesor jest przystosowany do adresowania 64—128 KB). Zakłada się możliwość przyłączenia typowych urządzeń zewnętrznych oraz wykorzystanie do sterowania procesami technologicznymi.

Terminal bankowy K 8924, oparty na mikroprocesorze K 1520, wyposażony jest w: monitor ekranowy o pojemności 1024 znaków z klawiaturą, pamięć na dyskach elastycznych, pamięć taśmową kasetową oraz drukarkę mozaikową (z możliwością stosowania kart kredytowych). Przewiduje się również możliwość dokonywania wpisów do książeczek oszczędnościowych oraz transmisji danych (9600 bitów/s).

Ekspozycja Bułgarii

Była stosunkowo skromna i obejmowała systemy ISOT 1003 C, oparte na 8-bitowym mikroprocesorze z pamięcią adresowalną do 64 KB (w tym stałą programowaną 19 KB). Wyposażone są w monitor ekranowy z klawiaturą alfanumeryczną, drukarkę o szybkości 30 znaków/s, 2—3 dyski elastyczne oraz modem dostosowany do współpracy z komputerem typu RIAD.

[1]) Awizowany był dalszy model z tej serii

Ekspozycja ZSRR

Wydawała się również skromna, mimo wystawienia rozbudowanego zestawu EC 7920 nazywanego systemem wielomonitorowym. Zaprezentowano jednostkę sterującą zestawem do 32 terminali (monitorów i drukarek znakowych). Zestaw ten może współpracować lokalnie lub zdalnie z komputerami RIAD.

*

Na tle stosunkowo bogatej ekspozycji niektórych krajów RWPG obecność firm zachodnich była bardzo skromna. Firma IBM wystawiła jedynie SYSTEM 1 oraz urządzenie końcowe systemu VIDEOTEXT (PRESTEL, TELETEL). Firma REDIFFISION była oblegana przez młodzież szkolną dzięki kolorowej grze komputerowo-telewizyjnej.

W tej sytuacji stosunkowo bogata była ekspozycja firmy DATA POINT, która przedstawiła dwa kompletne zestawy minikomputerowe: DP 1500 oraz DP 8800, który jest całkowicie nowym modelem.

DP 1500 służyć może jako mały system do zastosowań autonomicznych (w przetwarzaniu rozproszonym) lub jako inteligentny terminal. Jest wyposażony w pamięć operacyjną 4 KB ROM i 32—64 KB RAM (dostępną dla użytkownika), monitory ekranowe z klawiaturami oraz 2—4 dyski elastyczne (łączna pojemność 1 MB). Użytkownik może korzystać z następujących języków programowania: ASSEMBLER, DATABUS, DATAFORM.

DP 8800 jest najnowszą serią systemów minikomputerowych. Charakteryzuje się procesorem wieloprogramowym, pamięcią operacyjną do 1 MB, procesorem urządzeń peryferyjnych oraz modułami sprzęgającymi. Urządzeniami peryferyjnymi mogą być:

- 24 monitory ekranowe 1920-znakowe
- pamięci dyskowe 10, 20, 30, 60 a nawet większe (o łącznej pojemności do 1012 MB)
- drukarki (w tym laserowa)
- adaptery komunikacyjne
- pamięci taśmowe.

System może pracować w konfiguracji wieloprocesorowej (podobnie jak znane systemy DP 600-ARC). Przewiduje się dwa podstawowe sposoby wykorzystania: jako system „źródłowy", obsługujący zbiory danych oraz przetwarzanie partiowe, lub jako system „aplikacyjny", obsługujący sieć monitorów ekranowych w trybie interaktywnym.

DP 8800 jest wyposażony w system operacyjny RMS oraz następujące języki programowania: COBOL, DATABUS, DATAHARE.

W sieci ARC (procesor z serii 6000) można przyłączyć zestaw do prezentacji graficznej informacji barwnej (Color Business Graphics System) złożony z:

— kolorowego monitora ekranowego
— tablicy graficznej X, Y
— drukarki wielobarwnej
— urządzenia do wywoływania kolorowego filmu.

* * *

Nie sposób zaprezentować wszystkich urządzeń informatycznych wystawionych w Brnie. Nie pozwalają na to szczupłe ramy artykułu. Niektóre systemy, szczególnie produkcji CSRS, zasługują na oddzielne opracowanie. Ocena wystawianego sprzętu dokonana została z punktu widzenia potencjalnego użytkownika, a nie pod kątem nowinek konstrukcyjno-technologicznych.

Trudno oszacować na ile wystawa targowa miała względy prestiżowe, a na ile była realną ofertą handlową (informacji takiej nie można uzyskać). Skądinąd wiadomo, że CSRS zainteresowana jest szerokim eksportem sprzętu minikomputerowego i urządzeń peryferyjnych do Polski, szczególnie systemów SM 4-20 oraz w przyszłości — SM 52/11, a także autokreślarek i grafoskopów. Jest to w obecnej sytuacji jedyna realna szansa zdobycia sprzętu średniej klasy europejskiej. Być może wracamy do sytuacji z lat sześćdziesiątych, kiedy sprowadzaliśmy z CSRS, NRD i ZSRR maszyny systemu kart dziurkowanych — nie mając żadnej produkcji środków technicznych małej, średniej i wielkiej mechanizacji obliczeń.

JERZY SUKIENNIK

Męski świat

Skomputeryzowany świat jest światem mężczyzn. Poza nielicznymi wyjątkami, mężczyźni wymyślają gry video („pisane przez chłopców dla innych chłopców"), tworzą oprogramowanie, sprzedają sprzęt, uczestniczą w kursach informatycznych, gdzie średnio na ośmiu chłopców przypada jedna dziewczyna. Jeżeli taka sytuacja będzie trwać nadal, zwiększając różnice w umiejętnościach, sprawności i pewności siebie, współczesne dziewczyny niewątpliwie staną się szybko „obywatelami drugiej kategorii" — stwierdzają autorzy artykułu zamieszczonego w amerykańskim piśmie PSYCHOLOGY TODAY.

Dzieci poznają świat komputerów przede wszystkim dzięki grom video w salonach gry, w domach lub w lokalnych centrach komputerowych. Badania młodzieży posiadającej komputery w domach wykazały, że 88 proc. dzieci poniżej 12 lat i 67 proc. powyżej 12 większość czasu spędzonego z maszyną przeznacza na gry. Oprogramowanie komputerów domowych proponuje całe zestawy gier strategicznych, wojen gwiezdnych, bitew morskich i innych zabaw odpowiadających typowo męskim sportom. Salony gier video są miejscem spotkań młodych chłopców. Czasem towarzyszą im dziewczęta, które jednak o wiele częściej zajmują się podziwianiem swoich kolegów niż grą. Również sklepy ze sprzętem informatycznym są obcym miejscem dla większości kobiet i dziewcząt, nie oswojonych z elektroniką, drutami, układami, obwodami. Sklep z komputerami jest w rzeczywistości sklepem ze sprzętem elektronicznym. Nic zatem dziwnego, że pierwszymi klientami takich placówek byli mężczyźni-hobbyści w stylu tych, którzy dawniej sami kompletowali własne zestawy stereofoniczne. Operatorzy i sprzedawcy to przede wszystkim mężczyźni, najczęściej młodzi i żarliwie broniący — przed męskimi klientami — tezy, że używanie komputera określa sposób życia.

Oprogramowanie wykorzystywane w szkołach również odpowiada zainteresowaniom chłopców. Większość „gier edukacyjnych" dotyczy w takim samym stopniu wojen, przemocy, zespołowych gier w piłkę, jak standardowe propozycje salonów gier video.

Autorzy artykułu sugerują dwie możliwości zmiany opisanej sytuacji. Pierwsza — to tworzenie języków programowania, takich jak LOGO, którym może się posługiwać już czteroletnie dziecko. Różnice umiejętności i chęci współpracy z komputerem między chłopcami i dziewczynkami nie będą występować, jeżeli każde dziecko będzie mogło bawić się z maszyną w sposób, który najbardziej mu odpowiada — grać, rysować, pisać czy komponować. Druga — to zwracanie większej uwagi na sposób wykorzystania sieci informatycznych. Sama zawartość informacyjna sieci niewiele ma wspólnego z komputerem jako takim. Może dotyczyć przepisów kulinarnych, gwiazd piosenki, paleontologii — wszystkiego o czym zechce rozmawiać nastolatek. Ważne jest uświadomienie mu, że może wykorzystywać komputer tak jak chce. Szansę tę powinno się dać wszystkim dzieciom, jeżeli mają mieć równy start w komputerowym świecie przyszłości.

Angielski tygodnik NEW SCIENTIST również niezbyt optymistycznie pisze o miejscu kobiety w zinformatyzowanym społeczeństwie. Kobiety staną się ofiarami rewolucji w mikroelektronice. Pojawienie się komputerów i zautomatyzowanych maszyn

w życiu przemysłowym znowu zepchnie płeć piękną na sam dół drabiny społecznej. Taki jest przynajmniej pogląd trzech naukowców (kobiet — żeby nie było wątpliwości) z Technical Change Center w Londynie i Uniwersytetu Sussex. Kobiety wykonują prace, które już niedługo mogą przejąć komputery. Robotnice najprawdopodobniej stanowią większość zatrudnionych na liniach produkcyjnych, gdzie ludzie powinni być zastąpieni przez automaty). Również procesory tekstowe, (ang. word processor) mogą pozbawić pracy maszynistki.

„Na rynku pracy obowiązuje segregacja według płci, a kobiety stanowią większość zatrudnionych w pewnych gałęziach przemysłu i przy niektórych czynnościach. Wykonują typowe, powtarzalne, nudne zadania, nie wymagające specjalnych kwalifikacji, podobnie jak zajmowanie się gospodarstwem domowym" — oświadczyły wspomniane badaczki.

Pewne kategorie urzędników niewątpliwie zyskają na pojawieniu się komputerów. Dzięki procesorom tekstowym i ułatwieniem w pisaniu na maszynie, ich praca będzie bardziej interesująca. Więcej czasu pozostanie na inne zajęcia niż nudne przepisywanie listów czy narady. „Ale nie będzie to udziałem wszystkich urzędników — twierdzą dalej wspomniani naukowcy — wiele kobiet będzie godzinami wpatrywało się w ekrany monitorów. Trudno stwierdzić, na czym polega tu postęp w stosunku do tradycyjnej pracy maszynistki".

Komputery zagrożą w podobny sposób asystentkom sprzedawców w sklepach, pracownicom firm wysyłkowych i banków. Co więcej, kiedy kobiety będą chciały konkurować z mężczyznami w dziedzinach, gdzie niezbędna jest znajomość mikroelektroniki i informatyki, okaże się, że i tak nie mają szans, bo tradycyjne bariery społeczne zniechęcą je do zdobywania takich umiejętności.

Cytowane wyżej panie nie zostawiają również suchej nitki na tych, którzy twierdzą, że końcówki sieci komputerowej w domach pozwolą kobietom na łączenie pracy w gospodarstwie domowym z „chałupnictwem" w rodzaju pisania programów. Takie kobiety będą po prostu wykorzystywane finansowo przez pracodawców.

(ki)

TERMINOLOGIA

Terminologia grafiki komputerowej

Przedstawiony poniżej zestaw pojęć używanych w grafice komputerowej pokrywa się w zasadzie ze słownikiem ACM/SIGGRAPH (Graphics Glossary Committee) i z częścią dotyczącą grafiki komputerowej dokumentu ISO/TC97/SC1N (Computer Graphics and Computer Micrographics).

Grafika komputerowa (ang. computer graphics) obejmuje metody i techniki tworzenia, przedstawiania, przechowywania i przekształcania rysunków za pomocą komputerów. Rysunek jest tu najczęściej konturowym przedstawieniem pewnego modelu i składa się z **elementów rysunku** (ang. display elements), takich jak np. punkt, wektor, znak, tworzonych za pomocą **elementarnych operacji graficznych** (ang. graphic primitives). Dodatkowo — w strukturze rysunku mogą być wyróżniane **segmenty** (ang. display groups); tworzą one zespoły elementów rysunku, które mogą być modyfikowane łącznie, niezależnie od pozostałej części rysunku.

Do opisu struktury rysunku używany jest przeważnie kartezjański układ współrzędnych, przy czym w grafice komputerowej rozróżniane są trzy rodzaje współrzędnych: **współrzędne użytkownika** (ang. user coordinates) — tzn. te, którymi posługuje się użytkownik w konkretnych zastosowaniach, **współrzędne przestrzenne** (ang. world coordinate system) — służące do definiowania modeli w przestrzeni trójwymiarowej i **współrzędne urządzenia graficznego** (ang. device coordinate space) — używane do pozycjonowania mechanizmu kreślącego rysunek.

Współrzędne kolejnych punktów tworzonego rysunku mogą być podawane w sposób **bezwzględny** (ang. absolute) przy ustalonym początku układu współrzędnych lub w sposób **przyrostowy** (ang. relative, incremental), wówczas punktem odniesienia jest punkt poprzedni. Określenia te mogą być używane do opisu **danych graficznych** (ang. display data) lub **rozkazów graficznych** (ang. display orders, display commands).

Do przedstawiania rysunków służą **urządzenia graficzne** (ang. display devices); w zależności od ich konstrukcji rysunek jest kreślony lub wyświetlany. Do wyświetlania rysunków używane są: **ekrany plazmowe** (ang. plasma panels), **lampy elektronopromieniowe** (ang. cathode ray tubes) lub **lampy pamięciowe** (ang. storage tubes).

Obraz formowany za pomocą lampy pamięciowej istnieje na ekranie przez dłuższy czas bez konieczności jego **regeneracji** (ang. regeneration). Obraz formowany za pomocą lampy elektronopromieniowej jest cyklicznie powtarzany, przy czym **cykl wyświetlania** (ang. display cycle) musi odbywać się z dostatecznie dużą częstotliwością, gdyż w przeciwnym razie następuje **migotanie obrazu** (ang. flicker). Jasność niektórych elementów wyświetlanego rysunku może być zmieniana celowo w zauważalny sposób — operacja ta nazywana jest **pulsowaniem** (ang. blinking).

Do kreślenia rysunków służą **pisaki xy** albo **plotery** (ang. plotters); w zależności od ich konstrukcji rozróżniamy **pisaki stołowe** (ang. flatbed plotters) lub **bębnowe** (ang. drum plotters). Rysunek wykreślony za pomocą pisaka xy ma **trwałą postać** (ang. hard copy), natomiast wyświetlony za pomocą lampy elektronopromieniowej ma **nietrwałą postać** (ang. soft copy). Do sporządzania trwałych postaci rysunków wyświetlanych na ekranie lampy elektronopromieniowej służą **urządzenia kopiujące** (ang. hard copy devices).

W urządzeniu graficznym rysunek jest formowany na **osnowie** (ang. display surface), którą może stanowić **siatka punktów** (ang. dot matrix) lub **raster** (ang. raster). Rysunek formuje się na osnowie według ustalonego schematu, niezależnie od treści, w **rastrowych urządzeniach graficznych** (ang. raster displays), takich jak np. drukarki elektrostatyczne lub monitory telewizyjne. W **wektorowych urządzeniach graficznych** (ang. line-plotting displays), takich jak pisaki xy czy **grafoskopy** (ang. vector displays), rysunek jest kreślony lub wyświetlony wzdłuż linii tworzących jego strukturę.

Rozdzielczość (ang. resolution) urządzenia graficznego charakteryzuje zdolność zobrazowania szczegółów rysunku. Jednym z parametrów określających dokładność pisaka xy jest **wielkość kroku** (ang. plotter step size). W monitorach graficznych takim parametrem jest **liczba adresowalnych punktów osnowy** (ang. adressability measure).

Urządzenie graficzne może zawierać **generator wektorów/linii** (ang. vector generator), **generator znaków** (ang.

character generator) lub **generator krzywych** (ang. curve generator), które umożliwiają sprzętową realizację odpowiednich funkcji.

Grafika komputerowa, która uwzględnia możliwość bezpośredniego współdziałania użytkownika z systemem graficznym nazywana jest **interakcyjną grafiką komputerową** (ang. interactive computer graphics) w przeciwieństwie do **biernej grafiki komputerowej** (ang. passive computer graphics). Określenia te używane są również do opisu **trybu** (ang. mode) pracy urządzenia graficznego.

Jednym z technicznych uwarunkowań interakcyjnej grafiki komputerowej jest mechanizm lokalizowania poszczególnych elementów wyświetlanego rysunku; służy do tego celu **znacznik** (ang. cursor) ustawiany w odpowiednim miejscu na ekranie za pomocą **graficznego urządzenia wejściowego** (ang. graphical input device), takiego jak np. **manipulator kulowy** (ang. track ball), **manipulator drążkowy** (ang. joy stick), **klawiatura funkcyjna** (ang. function keyboard). Podobną funkcję może pełnić **pióro świetlne** (ang. light pen), przy czym do **pozycjonowania** (ang. positioning) za pomocą pióra świetlnego wymagany jest dodatkowy **znacznik pozycjonujący** (ang. aiming symbol). Do sterowania położeniem znacznika na ekranie może być używany **czytnik rysunków** (ang. digitizer).

Wyświetlany rysunek może być poddawany różnego rodzaju transformacjom, takim jak **skalowanie** (ang. scaling), **powiększanie** (ang. zooming), **przesuwanie** (ang. scrolling), **kadrowanie** (ang. windowing), tzn. wyświetlanie części rysunku znajdującej się wewnątrz zadanego **kadru** (ang. window). Operacja usuwania elementów rysunku leżących poza kadrem nazywana jest **obcinaniem** (ang. scissoring, clipping).

JACEK OWCZARCZYK

Kto pokocha sieci?

Budowa złożonych systemów informatycznych, takich jak sieć komputerowa, jest przedsięwzięciem bardzo trudnym. Wymaga właściwej synchronizacji wielu działań. A można ją realizować metodą „od dołu" (ang. bottom-up) lub „od góry" (ang. top-down)...

Metoda top-down jest stosowana w świecie często, szczególnie w USA. Gdzieś wysoko znajdują się osoby, które definiują zadanie, dzielą je na podzadania, przekazują do realizacji określonym zespołom, a następnie koordynują ich pracę. Taką metodą firma TYMSHARE w krótkim czasie zbudowała sieć TYMNET, monopolizując na długi czas usługi transmisji danych w świecie. Tak też powstała francuska sieć TRANSPAC. Ale zachwycając się tymi osiągnięciami często zapominamy, że celem podstawowym działalności organizacji budujących sieci jest zysk.

U nas dla budowy sieci komputerowej wyznaczono szczytne cele: zapewnienie pracownikom nauki wygodnego dostępu do zasobów obliczeniowych w kraju i za granicą, wyszkolenie kadry informatyków o różnych specjalnościach technologii sieciowej, opracowanie i zbadanie metod budowy i eksploatacji sieci komputerowych. I także zastosowano metodę top-down.

Prace nad sieciami komputerowymi uruchomiły — przypomnijmy — prawie równocześnie ZSRR, PRL, NRD, CSRS i WRL przy czym nie wyznaczono norm na współpracę węzłów tych sieci, ani też nie określono mechanizmu do wypracowania takich norm. W sieci RWPG będziemy więc musieli stosować węzły międzysieciowe lub inne dodatkowe mechanizmy dla wymiany informacji pomiędzy poszczególnymi krajami, zwiększające w istotny sposób koszty eksploatacji i pogarszające jakość usług.

Zadanie budowy węzłów podsieci komunikacyjnej zostało w Polsce powierzone różnym zespołom, przy czym również tutaj nie określono zasad współpracy między budowanymi węzłami. W tym przypadku wczesna kontrakcja tych zespołów dała pozytywny skutek: uzgodniono i wdrożono zasady współpracy węzłów, chociaż dotąd nie zostały one formalnie zatwierdzone.

Dotychczas nie zaakceptowano jednak udziału żadnego użytkownika, który sprawdzałby całe nitki w sieci, mimo że liczny zespół fachowców z wielu zaangażowanych instytucji uznał taką konieczność. Jest to w całkowitej zgodzie z teorią Parkinsona, według której system na pewnym stopniu rozwoju nie potrzebuje jakiegokolwiek otoczenia i może dobrze rozwijać się sam.

Przy niepełnym doinformowaniu zapadają na wysokim szczeblu, decyzje błędne, które powodują poważne i długotrwałe skutki (np. realizacja usług sieciowych na R-32 i opóźnienie prac na ODRZE 1305). Występuje to niestety bardzo często z uwagi na powszechny brak pogłębionej analizy sytuacji, dobrego warsztatu badawczego czy po prostu — brak kompetencji i odpowiednich kwalifikacji.

Zespoły realizujące sieci w Polsce są od siebie całkowicie odizolowane (często w wyniku świadomego działania w myśl zasady „divide et impera") i działają samozachowawczo. Jak żart brzmi informacja, że ludzie otrzymujący takie samo zadanie, choć na innym sprzęcie, poznają się dopiero na konferencjach krajowych lub zagranicznych.

Jestem przekonany, że metoda top-down dla tak złożonego i nowatorskiego przedsięwzięcia, jak budowa sieci komputerowej jest u nas nie do przyjęcia. Doświadczenia należy bowiem zdobywać mozolnie, drogą indukcyjną, a słuszność decyzji i rozwiązań sprawdzać na każdym możliwym etapie, stosując maksymalnie obiektywne kryteria.

Drogą taką szli Amerykanie przy budowie sieci ARPANET oraz Francuzi budując CYCLADES. Podobnie postępuje Poczta Brytyjska, która rozpoczęła prace od modemów (DATEL) dla systemów wielodostępnych i poprzez sieć eksperymentalną EPSS doszła do bardzo udanej — z punktu widzenia ponoszonych przez użytkowników kosztów — sieci PSS.

Z krajów RWPG postępuje tak Bułgaria. Na własnych maszynach Jednolitego Systemu z wykorzystaniem systemu operacyjnego DOS+CICS zbudowała na prostej podsieci datagramowej system informowania kierownictwa dużych zakładów przemysłowych, który jest eksploatowany już od kilku lat. Bułgarzy wyszli z założenia, że nie stać ich na niesprawdzalne eksperymenty, ponieważ kraj jest zbyt biedny.

Obyśmy wreszcie skorzystali z tych doświadczeń i podjęli zdecydowane i wytrwałe działania: od góry do dołu — przez obiektywizację kryteriów oceny prac i zapewnienie zgodności celów rzeczywistych z teoretycznymi, oraz od dołu do góry — przez tworzenie zgranych, formalnych lub nieformalnych zespołów, które będą zmierzały do jednego celu: zaspokojenia potrzeb użytkownika. Tak złożonych systemów nie można tworzyć w sytuacji wzajemnej izolacji zespołów projektujących, tolerowania ich partykularnych interesów i preferowania asekuracji zamiast pełnej odpowiedzialności za całość zagadnienia (mimo że realizują one tylko jego fragmenty) oraz komfortowego sterowania przedsięwzięciem metodą top-down.

W przeciwnym razie nieprędko ktokolwiek spoza zespołów realizujących sieci komputerowe zrozumie ich niezbędność!

JÓZEF B. LEWOC

Abramowicz W.: MODULA-2 i LILITH — zgodność metod i narzędzi informatycznych

INFORMATYKA 1984, nr 4, s. 1

Charakterystyka rozwiązań komputera osobistego LILITH oraz języka programowania MODULA-2, opracowanych przez prof. N. Wirtha, a stanowiących przykład konsekwentnej synchronizacji możliwości sprzętu z jego oprogramowaniem.

Абрамович В.: MODULA-2 и LILITH — соответствие методов вычислительной техники и аппартных средств

INFORMATYKA 1984, № 4, стр. 1

Характеристика решений личной ЭВМ LILITH и языка программирования MODUL-2, разработанных проф. Н. Виртом и представляющих собой пример последовательной синхронизации возможностей аппаратуры с ее программным обеспечением.

Kott R. K., Magdzik D.: Techniki interpretacji dla mikrokomputerów

INFORMATYKA 1934, nr 4, s. 6

Charakterystyka współczesnych metod interpretacji stosowanych przy eksploatacji mikrokomputerów. Omówiono i porównano z interpretacją klasyczną parametry eksploatacyjne oraz koszty czterech nowych technik interpretacji.

Котт Р. К., Магдзик Д.: Техники интерпретации для микро-ЭВМ

INFORMATYKA 1984, № 4, стр. 6

Характеристика современных методов интерпретации применяемых в эксплуатации микро-ЭВМ. Обсуждено и сравнено с классической интерпретацией эксплуатационные парамтеры и издержки четырех новых техник интерпретации.

Sobczyk M., Szałas A.: PROLOG (2)

INFORMATYKA 1984, nr 4, s. 10

Dokończenie ogólnej charakterystyki języka PROLOG. Podano przykłady ułatwiające zrozumienie jego podstawowych mechanizmów.

Собчик М., Шалас А.: PROLOG (2)

INFORMATYKA 1984, № 4, стр. 10

Завершение общей характеристики языка PROLOG. Данны примеры облегчающие понимание его основных механизмов.

Abramowicz W.: MODULA-2 and LILITH — computing methods and tools compatibility

INFORMATYKA 1934, No. 4, p. 1

Characteristics of LILITH personal computer and MODULA-2 programming language solutions, elaborated by prof. N. Wirth, as an example of consequent synchronization of hardware possiblities with its software.

Abramowicz W.: MODULA-2 und LILITH — Einklang von Informatikmethoden und -Werkzeuge

INFORMATYKA 1984, Nr. 4, S. 1

Eine Charakteristik des LILITH-Personalcomputers und der MODULA-2 — Programmiersprache, die von Prof. N. Wirth erarbeitet wurden und ein Beispiel der konsequenten Synchronisierung von Hardwaremöglichkeiten mit ihrer Software darstellen.

Kott R. K., Magdzik D.: Interpretation technics for microcomputers

INFORMATYKA 1934, No. 4, p. 6

Characteristics of contemporary interpretation methods applied for microcomputers. Operation parameters and costs of four new interpretation technics are discussed and with classic interpretation compared.

Kott R. K., Magdzik D.: Interpretierungstechniken für Mikrorechner

INFORMATYKA 1984, Nr. 4, S. 6

Eine Charakteristik von modernen Interpretierungsmethoden, die für Mikrorechner ausgenutzt werden. Es werden Betriebsparameter und Kosten von vier neuen Interpretierungstechniken besprochen und mit klassischer Interpretierung verglichen.

Sobczyk M., Szałas A.: PROLOG (2)

INFORMATYKA 1984, No. 4, p. 10

Termination of the PROLOG general characteristics. Examples, which facilitates understanding of its basic mechanisms, are presented.

Sobczyk M., Szałas A.: PROLOG (2)

INFORMATYKA 1984, Nr. 4, S. 10

Beendigung der allgemeinen Charakteristik von PROLOG-Programmiersprache. Es werden Beispiele, die bessere Verständigung ihrer Grundmechanismen erleichtern, angegeben.

■ W 1982 roku amerykańska firma SHUGART, która zapoczątkowała stosowanie dyskietek do komputerów, zaplanowała przy współpracy z francuską firmą THOMSON CSF wprowadzenie 30-centymetrowych dysków z odczytem optycznym (laserowym). Jedna strona takiego dysku może pomieścić 1000 M bajtów informacji, co odpowiada czterystu tysiącom stron tekstu. Koszt dysku wraz z urządzeniem odczytującym, a także przyłączenia go do komputera — wynosić będzie 8—10 tys. funtów. Tymczasem firma STORAGE TECHNOLOGY produkująca duże dyski, proponuje za 140 tys. funtów system dysków z odczytem optycznym o czterokrotnie większej pojemności, dostosowany do współpracy z dużymi komputerami IBM. Jak wiadomo, komputery wymagają o wiele większej precyzji odtwarzania informacji niż inne urządzenia elektroniczne. O ile błąd zapisu rzędu 1 na 10^6 przetwarzanych bitów jest niedostrzegalny na ekranie telewizora, to maszyna cyfrowa toleruje poziom błędu co najwyżej 1 na 10^{12} bitów. Problemy te idealnie rozwiązuje system z odczytem laserowym. Ma on jednak niezwykle istotną wadę — brak możliwości łatwej wymiany zapisanych informacji. Praktycznie więc dyski z odczytem laserowym mogą być stosowane tylko do przechowywania danych stałych. Żywotność dysku określa się na co najmniej dziesięć lat. Japońska firma MATSUSHITA zapowiada wyprodukowanie dysku z odczytem optycznym, umożliwiającego wielokrotny odczyt i zapis — jak w przypadku dysków i taśm magnetycznych. Ale na razie są to tylko obietnice. (K)

*

■ Globalny model atmosfery, opracowany przez Europejskie Centrum Średnioterminowych Prognoz Pogody (organizacja skupiająca 17 krajów europejskich), pozwolił uzyskać dane, na podstawie których wykreślono rozkład chmur na kuli ziemskiej. Pomiary takich wielkości atmosferycznych, jak temperatura, ciśnienie prędkość wiatru oraz wilgotność — umożliwiają, po ich uśrednieniu, opasanie całej kuli ziemskiej siecią odpowiednich wartości. Model atmosfery oblicza przyszłe warunki atmosferyczne, wynikające z wartości uśrednionych, poprzez rozwiązanie układu równań różniczkowych (newtonowskie prawa ruchu, równania termodynamiczne oraz równanie gazu). Rozdzielczość uyskiwanego obrazu pozwala uzmysłowić sobie, jak wiele potrzeba obliczeń do modelowania rozwoju sytuacji w czasie. Dla przykładu — komputer Centrum Prognoz Średnioterminowych CRAY-1 musi wykonać 500 mld operacji arytmetycznych, by przewidzieć zachowanie się atmosfery po dziesięciu dniach. Ponieważ CRAY-1 wykonuje ok. 80 mln operacji arytmetycznych na sekundę to powyższe obliczenia trwają ok. 6250 s, tzn. 1 godzinę i 45 minut. (T)

*

■ Japońscy inżynierowie skonstruowali urządzenie, które stokrotnie przyspieszyło graficzne przedstawianie danych przesyłanych na Ziemię przez satelity. Dane takie zawierają wielką liczbę „surowych" informacji, którym dotychczas nadawano czytelną formę dopiero po kilkudniowej pracy komputera. Stosując nowatorskie połączenie dwóch technik obliczeniowych — przetwarzania równoległego i łączenia procesów (ang. pipelining) — Japończycy skrócili do 3,5 godz. czas tworzenia zdjęcia satelitarnego. Konwencjonalne rozwiązania zakładały pobieranie każdego bitu informacji z pamięci komputera, przetwarzanie i ponowne przesłanie do pamięci, co zwiększało czas pracy, a przy tworzeniu obrazów — również obszar wykorzystywanej pamięci. Nowa metoda eliminuje przesłanie pamięć-procesor-pamięć, ponieważ wynik operacji stanowi wejście do wykonania następnej. Szybkość przetwarzania wynosi ok. 53 mln operacji na sekundę przy stosunkowo prostym oprogramowaniu. Zwiększając liczbę procesorów można uzyskać jeszcze lepsze rezultaty. (K)

*

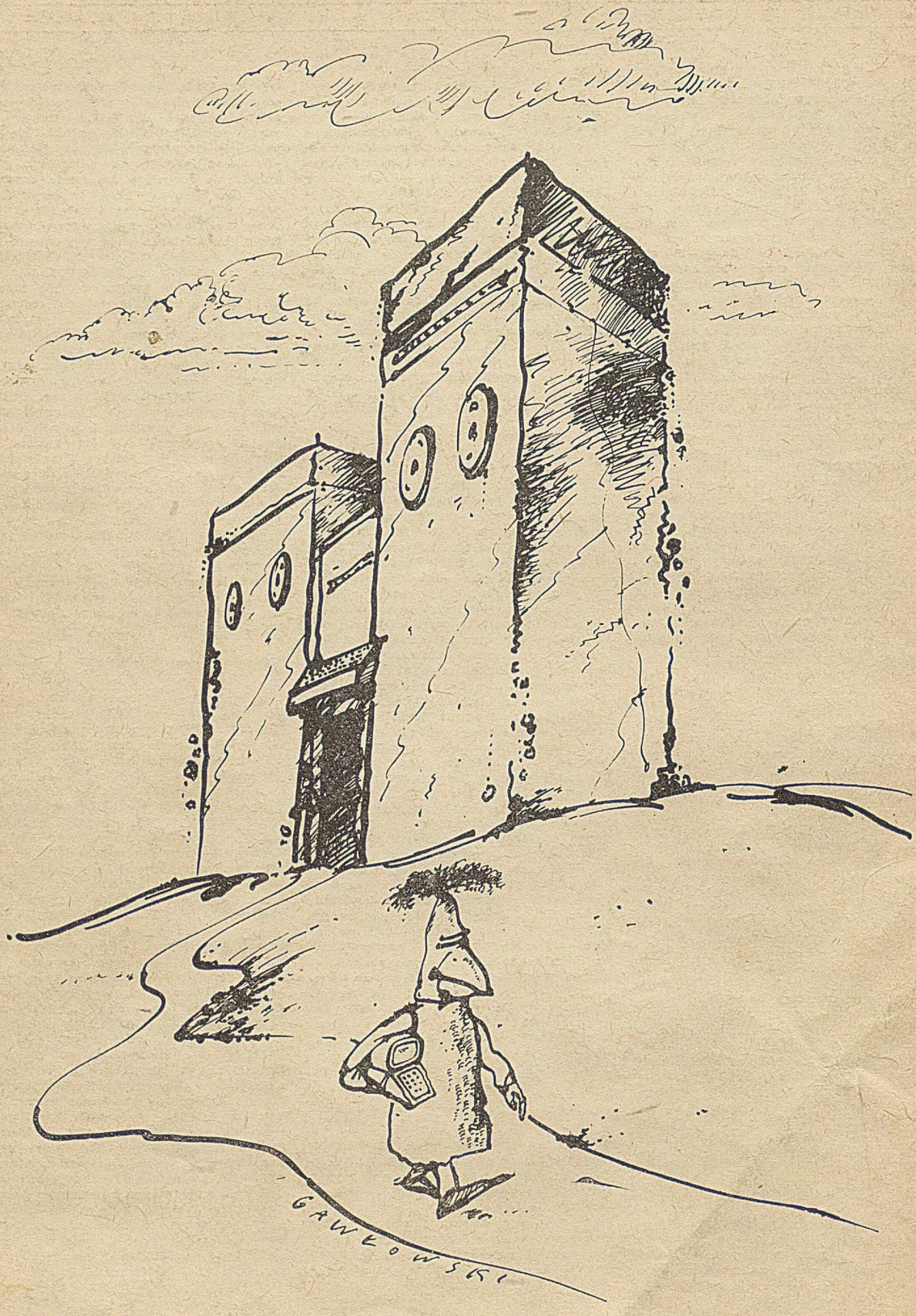
■ Przewidywanie struktury kryształów za pomocą komputera osiągnęło tak dużą niezawodność, że mineralodzy w pełni docenili jego zalety. S. C. Parker z grupą uczonych londyńskiego University College (UCL) określili dokładnie budowę dwóch ważnych krzemianów posługując się własną „metodą minimalizacji energii". Nauki chemiczne korzystają więc coraz szerzej z osiągnięć informatyki, wzrastającej mocy obliczeniowej maszyn cyfrowych i udoskonalania technik programowania .Wolni od dawnych ograniczeń, naukowcy dysponują obecnie narzędziem do określania struktur o dużej liczbie atomów. Kolejny krok — przewidywanie struktury kryształu — był nielatwy, ale Parker wykonał go definiując sposób oddziaływania ładunku danego jonu na jony sąsiednie. Opracowana przez Parkera metoda obliczania energii potencjalnej między dwoma jonami opiera się na założeniu, że elektrostatyczne oddziaływania jonów sąsiednich na dany jon mogą być rozpatrywane łącznie. Sprowadza to rozważania dotyczące wielu elementów do rozwiązania problemu oddziaływania dwóch ciał. Zespół z UCL wykorzystał komputer do znalezienia takiego położenia jonów, by — znając siły ich wzajemnego przyciągania i odpychania — otrzymać układ najbardziej stabilny energetycznie. Technika minimalizacji energii pozwala określać warunki najtrwalszego wzajemnego położenia jonów dodatnich i ujemnych. Uzyskane wyniki w pełni potwierdzają dane empiryczne. Również pewne osobliwości zachowania atomów tlenu zostały nieoczekiwanie wykazane przez program dokonujący obliczeń. (K)

*

■ Jesienią ubiegłego roku Uniwersytet Surrey rozpoczął prace nad badaniem skutków używania lekarstw przepisywanych przez lekarzy pięciu milionom Brytyjczyków. Komputerowy system zbierania informacji będzie szybszy i efektywniejszy niż metody tradycyjne i powinien zapewnić lepsze rozpoznanie nieznanych dotąd efektów ubocznych zażywanych medykamentów. Wadą proponowanego rozwiązania może być trudność z zachowaniem poufności informacji wprowadzanych do pamięci maszyny. System opiera się na względnie nowym pomyśle „monitorowania zdarzeń towarzyszących leczeniu", gdzie opisuje się wszystkie „zdarzenia zdrowotne", nie tylko te, które w powszechnej praktyce lekarskiej uważa się za efekt zastosowanej terapii. Projekt przedstawiony przez uniwersytet zakłada wyposażenie 2500 lekarzy w sześciuset placówkach medycznych w terminale typu PRESTEL. Informacje uzyskane od pacjentów będą przesyłane liniami telefonicznymi do centralnego komputera. Planuje się w przyszłości udostępnienie systemu 10 tysiącom lekarzy, co pozwoli objąć kontrolą połowę brytyjskich pacjentów. Lekarze z coraz większą niechęcią odnoszą się do obowiązującego w Wielkiej Brytanii systemu tzw. Żółtych Kart — przekazywania ministerstwu zdrowia danych o ubocznych skutkach działania poszczególnych środków. System nie zapewnia uzyskania pełnych informacji na ten temat, a podstawową trudność stanowi konieczność każdorazowego rozstrzygania, czy dana reakcja organizmu pacjenta jest związana z przyjmowaniem określonego lekarstwa. Nowe rozwiązanie — dzięki zgromadzeniu wielkiej liczby danych — pozwala łatwiej i precyzyjniej definiować skutki zażywania współdziałających lekarstw. Oczekuje się, że koszty eksploatacji systemu, poza zainstalowaniem terminali, nie będą duże. (ki)

*

■ Rząd brytyjski przeznaczył 2,5 mln funtów na zakup urządzeń elektronicznych, głównie mikrokomputerów, dla siedmiuset szkół specjalnych. Minister ds. technologii informatycznej, Kenneth Baker, oświadczył, że „komputery pozwalają dzieciom upośledzonym sprawdzać się bez współdziałania z innymi ludźmi, w dziedzinach niedostępnych wielu dorosłym, zwiększają poczucie własnej wartości. Pojawiają się nowe, wspaniałe możliwości nauczania i przystosowania do życia w normalnym społeczeństwie". Minister stwierdził, że wiele firm obawia się podjęcia ryzyka związanego z produkcją sprzętu dla ludzi niepełnosprawnych, ponieważ jest ona pozornie mniej opłacalna. „Popełniają wielki błąd" — podsumował, podając przykłady sukcesu rynkowego płyt długogrających przeznaczonych wyłącznie dla upośledzonych i urządzeń odczytujących tekst niewidomym. Za pieniądze rządowe zakupi się taki sprzęt, jak „Żółw" — mały robot na kółkach sterowany mikrokomputerem, „Sound Bubble" — urządzenie pomagające kalekim dzieciom zwiększać koordynację ruchową, „Concept Keyboard" — klawiatura z wielkimi przyciskami, które mogą być oznaczane różnymi symbolami. Obecnie działa ok. 100 egzemplarzy każdego z tych urządzeń. Rząd zakupił również 500 maszyn VISTEL pozwalających głuchym „drukować" rozmowy telefoniczne. „Dzisiejsze ułatwienia dla niepełnosprawnych mogą być jutro produktami masowymi" — stwierdził minister Baker. (ki)


GAWLOWSKI