mikroKLAN 3

3 1984

informatyka

PROLOG
CAMAC
HP-IL

WYDAWNICTWO SIGMA

informatyka

3 1984

Nr 3
Miesięcznik Rok XIX
Marzec 1984

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

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 2025. Obj. 4,0 ark. druk. Nakład 3750 egz. T-46.

INDEKS 36124

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

NACZELNA ORGANIZACJA TECHNICZNA
WYDAWNICTWO
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

W NUMERZE: Strona

PROLOG (1)
Marek Sobczyk, Andrzej Szałas 1

Quo vadis, Jednolity Systemie?
Józef B. Lewoc 5

MODULA-2 — język lat osiemdziesiątych (3). Implementacja ETH
Piotr Fuglewicz 8

Laboratorium dydaktyczne w systemie CAMAC z rozłożoną inteligencją
Jacek Mirkowski, Adam Piątkowski 11

Analizator ATA-77 w systemie CAMAC do analizy dwuparametrycznej
Ryszard Gacek, Andrzej Gościński, Zbigniew Rempała, Roman Rygał 21

Przegląd języków wysokiego poziomu (5). Rozwój i tematy prac badawczych
Oprac. Halina Ciechomska, Teresa Wójciekian 23

mikroKLAN 13

— Człowiek z Perskiego
— Nieznane rozkazy mikroprocesora 8085
— Układ pracy krokowej dla Z80
— Przekazywanie sterowania do programu MONITOR
— FORTH (3)
— TEST

Z KRAJU 26

— Rada ds. Zastosowań Środków Techniki Obliczeniowej

ZE ŚWIATA 27

— HP-IL
— Piraci informatyki
— IFIP
— Paradoks ergonomii
— Wizja bio-kostki

TERMINOLOGIA 31

— Dokumentacja

POGLĄDY III okł.

— Komputer — rzecz prywatna — *Jakub Tatarkiewicz*

CZWARTA OKŁADKA — *Jacek Gawłowski*

W NAJBLIŻSZYCH NUMERACH:

- Roman Żelazny o narzędziach inżynierii oprogramowania
- Witold Abramowicz o MODULI-2 i LILITH
- Ryszard K. Kott i Danuta Magdzik o technikach interpretacji dla mikrokomputerów
- Cezary Zieliński i Teresa Wilczek o robotyce
- Tadeusz Szuba o związku PROLOGU z PASCALEM
- Wojciech Trojniar o języku i systemie programowania FORTH
- Tadeusz Mazurkiewicz o regresie polskiej informatyki

MAREK SOBCZYK
ANDRZEJ SZAŁAS
Warszawa

PROLOG (1)

Zacznijmy od usprawiedliwienia — dlaczego prezentujemy PROLOG na łamach INFORMATYKI, skoro na krajowym rynku księgarskim ukazał się niedawno szczegółowy opis tego języka [3]. Otóż istnieją dwie podstawowe przyczyny, dla których uznaliśmy, że trud taki wart jest podjęcia. Z jednej strony — wspomniana książka napisana została przed trzema laty, nie odzwierciedla więc w pełni aktualnego stanu prac nad PROLOGIEM. Po drugie — wpadł nam ostatnio w ręce artykuł [2], a zawarty w nim sposób prezentacji języka wydał się nam nader obrazowy. Toteż niniejsza, pierwsza część opisu PROLOGU bazuje głównie — co do koncepcji i doboru przykładów — na wspomnianej pracy [2].

PROLOG jest językiem programowania bardzo wysokiego poziomu. Został on stworzony przez A. Colmerauera i jego grupę z Uniwersytetu Marsylskiego. Podstawy PROLOGU wywodzą się z idei angielskiego logika i teoretyka informatyki — Roberta Kowalskiego, który postawił tezę, iż język logiki umożliwia specyfikowanie problemów informatycznych i ich programowanie w sposób najwygodniejszy dla człowieka. Toteż mechanizmy, jakie uzyskuje użytkownik PROLOGU, różnią się istotnie od mechanizmów FORTRANOWYCH czy PASCALOWYCH. O ile te ostatnie stanowią uproszczenie zasad programowania w języku maszyny, o tyle PROLOG posługuje się terminami niezależnymi od komputera; nie wprowadza np. pojęcia instrukcji czy kolejności ich wykonania, choć można w PROLOGU symulować takie konstrukcje, jak podstawienie, iteracja, rekurencja.

PROLOG jest w swej istocie językiem specyfikacyjnym. Wystarczy opisać w nim problem za pomocą twierdzeń i reguł wnioskowania, charakteryzujących właściwości przyjętych obiektów. Jeśli charakteryzacja ta jest dostatecznie dokładna maszyna potrafi rozwiązać zagadnienie bez dodatkowego udziału programisty. Tak więc użytkownik może poświęcić cały swój wysiłek na specyfikację problemu, zaś zadania podrzędne są brane pod uwagę i rozwiązywane przez maszynę dopiero w trakcie ich pojawienia się podczas interpretacji PROLOGU. Pozwala to na fundamentalnie niedeterministyczny sposób pracy z obiektami częściowo lub całkowicie nieznanymi, tak jak się to — na przykład — dzieje w matematyce podczas rozwiązywania równań.

Na początku PROLOG był narzędziem dowodzenia twierdzeń, opierającym się na metodzie A. Robinsona [5] i zawierał drakońskie ograniczenia zmniejszające obszar badań (dowodzenie liniowe, dostęp wyłącznie do pierwszego literału każdej formuły itd.). W pracy [4] R. Kowalski i M. Van Emden wykazali, że te ograniczenia są równoważne użyciu klauzul Horna, tzn. takich formuł, które mają co najwyżej jeden literał pozytywny. Stworzyli oni model teoretyczny, który Colmerauer wykorzystał do systematyzacji użycia gramatyk i uzyskania języka o tej samej mocy (w przetwarzaniu języków naturalnych) co system Q [1], który może być uważany za poprzednika PROLOGU.

Pierwszy interpreter został napisany w roku 1973 w laboratorium sztucznej inteligencji na Uniwersytecie Marsylskim przez P. Roussela. Korzystał on z zasady niekopiowania termów, a ostre ograniczenia (dowodzenie liniowe, uporządkowanie literałów formuły, sterowanie niedeterminizmem, unifikacja bez sprawdzania występowania zmiennej) uczyniły z PROLOGU użyteczny język programowania.

Od tego czasu PROLOG został rozpowszechniony w wielu krajach (we Francji, Wlk. Brytanii, Portugalii, Hiszpanii, USA, Kanadzie, Polsce [1]), na Węgrzech, itd.). Pośród wielu implementacji warto wymienić kompilator napisany przez D. Warrena na maszynę DEC-10. Później przeniesiono go na mikrokomputery (autorzy języka na EXORCISER, a F. Mc Cabe na SORCERER). W ostatnim czasie, w związku z japońskimi projektami maszyn piątej generacji, zainteresowanie PROLOGIEM wyraźnie wzrosło.

Język ten wykorzystywany jest w takich dziedzinach sztucznej inteligencji, jak: komunikacja z komputerem w języku naturalnym, obliczenia formalne, programowanie robotów, pisanie kompilatorów, banki danych, komputerowe wspomaganie projektowania, systemy typu ekspert, itd., itd.

W społeczności informatyków PROLOG osiągnął sukces dzięki pewnym dodatkom, przerażającym z punktu widzenia teorii, ale koniecznym dla umożliwienia praktycznego programowania w wymienionych dziedzinach. Przykładem może być tu słynny przypadek operatora „/", który to operator wydaje się być niezbędny, jeśli chce się ograniczyć liczbę dróg badanych przez interpreter.

PROLOG II

Programy PROLOGOWE bardzo szybko stały się duże i złożone. Ograniczenia dostępnych maszyn (jak np. rozmiar pamięci) szybko zostały przekroczone. W związku z tym autrzy języka opracowali nową jego wersję, biorąc pod uwagę następujące czynniki.

- możliwość przenoszenia oprogramowania dzięki stworzeniu maszyny wirtualnej obejmującej komputery wszystkich wielkości, łącznie z mikrokomputerami
- interakcyjną współpracę z PROLOGOWYM programem redagującym formuły
- modularność, uzyskaną dzięki zorganizowaniu obszaru formuł w hierarchiczne podzbiory, zbudowane jako drzewa, oraz dzięki wprowadzeniu rozkazów operowania tymi podzbiorami
- rozszerzalność, wynikłą z możliwości dołączania podprogramów zapisanych w języku konkretnej maszyny (arytmetyka, obsługa urządzeń peryferyjnych itp.), które traktuje się jako predykaty wyliczalne
- niezawodność, wyrażającą się w wymuszaniu prawidłowego przekazywania informacji o błędach.

W drugiej wersji PROLOGU pojawiły się także pewne nowe koncepcje. Przykładowo — wprowadzony został predykat **geler**, pozwalający opóźnić wykonanie procesu aż do momentu, w którym pewna informacja stanie się znana. Możliwe jest wyrażenie poprzez predykat **dif**, że dwa jeszcze nieznane drzewa są i pozostaną zawsze różne. Wprowadzono także przetwarzanie drzew nieskończonych. Do sterowania wykonaniem programów wykorzystano koncepcję bloku.

PROLOG jest językiem programowania przeznaczonym do reprezentowania wiedzy dotyczącej pewnej dziedziny i posługiwania się tą wiedzą. Dokładniej — dziedzina jest zbiorem obiektów, a wiedza wyraża się przez relacje opisujące właściwości tych obiektów i ich wzajemne związki. Zbiór reguł opisujących obiekty i relacje stanowi program w PROLOGU.

Przykładowo — zdanie: „Jan jest ojcem Pawła" jest określeniem relacji bycia ojcem, wiążącej dwa obiekty (opisane przez imiona Jan i Paweł), którą można zapisać:
jest-ojcem (Jan, Paweł).

[1]) Od kilku lat istnieje interpreter PROLOGU na ODRZE 1305 oraz na maszynie IBM-360, kompilator na CDC-6000, ostatnio ukończono prace nad implementacją PROLOGU na maszynie SM-4 i MERA 400.

Tak samo pytanie: „kto jest ojcem Pawła?" prowadzi do sprawdzenia, czy relacja jest-ojcem łączy Pawła z innym obiektem, stanowiącym odpowiedź na to pytanie. Zauważmy, że w definicjach relacji porządek, w jakim występują obiekty, jest znaczący: jest-ojcem (Jan, Paweł) różni się od jest-ojcem (Paweł, Jan).

W restauracji

Dla zilustrowania mechanizmów PROLOGOWYCH prześledźmy przykład, który opisuje kartę dań w restauracji. Interesującymi nas obiektami są dania, które można tam zamówić. Pierwsza seria relacji klasyfikuje je jako przystawki, dania mięsne lub rybne i desery. Karta, stanowiąca mały bank danych, będzie opisana następująco:

```
„Karta"                    (i)

przystawka (karczochy-Melania) → ;
przystawka (trufle) → ;
przystawka (jajka-sadzone-z-rukwią) → ;
mięso (wołowina-z-rusztu) → ;
mięso (kurczak-w-lipie) → ;
ryba (okoń-w-algach) → ;
ryba (sandacz-faszerowany) → ;
deser (lody-gruszkowe) → ;
deser (truskawki-z-Chantilly) → ;
deser (melon-niespodzianka) → ;
```

Zdefiniowane relacje wprowadzają obiekty i ich klasyfikację. Np. **przystawka(trufle) → ;** oznacza, że trufle są przystawką. Ten pierwszy typ reguł wyraża proste twierdzenia. Po zdefiniowaniu takiego zbioru twierdzeń można zadawać dotyczące go pytania. Pytanie typu „Czy trufle są przystawką?" zapisuje się w PROLOGU:

```
przystawka(trufle);
```

Następuje wtedy sprawdzenie, czy twierdzenie takie jest prawdziwe. W rozpatrywanym przypadku odpowiedź brzmi „tak". Natomiast pytanie **przystawka(sałatka-pomidorowa);** spowoduje odpowiedź negatywną, ponieważ bank nie zawiera takiego twierdzenia.

Przypuśćmy teraz, że chcemy wiedzieć jakie są przystawki. Byłoby niewygodne, a czasami wprost niemożliwe, zadanie po kolei pytań w formie dotychczas przedstawionej i oczekiwanie odpowiedzi „tak" albo „nie" w każdym przypadku. Lepiej byłoby zapytać: „jakie są przystawki?", a dokładniej — „jakie są obiekty **e**, które są przystawkami?". W tym przypadku **e** nie oznacza konkretnego obiektu, ale wszystkie obiekty mające właściwości przystawki. Mówimy wówczas, że **e** jest zmienną. W naszym przypadku pytanie brzmi:

```
przystawka (e);
```

i PROLOG odpowiada:

```
e = karczochy-Melania
e = trufle
e = jajka-sadzone-z-rukwią
```

wypisując zbiór obiektów, dla których twierdzenie **przystawka(e)** jest spełnione. W PROLOGU zmienne oznacza się za pomocą identyfikatorów zaczynających się literą, po której następuję (być może pusty) ciąg cyfr.

Na podstawie relacji tworzących początkowy bank danych można konstruować relacje bardziej złożone lub bardziej ogólne. Na przykład z relacji **mięso( )** i **ryba( )** określających, że argument jest daniem mięsnym lub rybnym, można zdefiniować relację **danie( )** jako danie mięsne lub rybne, co zapisuje się:

```
danie(p) → mięso(p);        (ii)
danie(p) → ryba(p);
```

Zapis ten odczytujemy jako „**p** jest daniem, o ile **p** jest daniem mięsnym; **p** jest daniem, o ile **p** jest daniem rybnym" (taka sekwencja dwóch reguł interpretowana jest jako alternatywa). Użyliśmy tu zmiennej **p**, która w każdej z dwóch reguł oznacza odpowiednio wszystkie dania mięsne i wszystkie dania rybne. Zakres zmiennej jest ograniczony do reguły, w której ją zdefiniowano, a więc zmienna **p** z pierwszej reguły nie jest związana ze zmienną **p** z drugiej.

W naszym przykładzie pytanie „jakie są dania?" wyrażone:

```
danie(p);
```

wywołuje odpowiedzi:

```
p = wołowina-z-rusztu
p = kurczak-w-lipie
p = okoń-w-algach
p = sandacz-faszerowany
```

Zajmijmy się teraz ułożeniem posiłku. Zgodnie ze zwyczajem, posiłek składa się z przystawki, dania głównego i deseru. Posiłek jest więc trójką **e, p, d**, gdzie **e** jest przystawką, **p** — daniem, a **d** — deserem. Wyrażamy to w sposób naturalny za pomocą reguły:

```
posiłek(e, p, d) → przystawka(e)  danie(p)  deser(d);  (iii)
```

którą czytamy: **e, p, d** stanowią posiłek, jeśli — **e** spełnia relację przystawki i **p** spełnia relację danie i **d** spełnia relację deser. Zdefiniowaliśmy w ten sposób nową relację jako koniunkcję trzech innych relacji. Na pytanie „jakie są posiłki?", tzn.

```
posiłek(e, p, d)
```

PROLOG odpowie wypisując listę trzydziestu sześciu możliwych kombinacji:

```
e = karczochy-Melania  p = wołowina-z-rusztu  d = lody-gruszkowe
e = karczochy-Melania  p = wołowina-z-rusztu  d = truskawki-z-Chantilly

.....

e = karczochy-Melania  p = sandacz-faszerowany  d = melon niespodzianka
e = trufle  p = wołowina-z-rusztu  d = lody-gruszkowe

.....

e = trufle  p-= sandacz-faszerowany  d = melon-niespodzianka
e = jajka-sadzon-z-rukwią  p = wołowina-z-rusztu  d = lody gruszkowe

.....

e = jajka-sadzone-z-rukwią  p = sandacz faszerowany  d = melon-niespodzianka
```

Zadajmy teraz nieco dokładniejsze pytanie dotyczące tego samego zbioru relacji — chcemy znać posiłki zawierające rybę jako danie główne. Zapisujemy takie pytanie jako:

```
posiłek(e, p, d) ryba(p);
```

co wyraża koniunkcję dwóch warunków, które mają być spełnione. Po wyliczeniu **posiłek ( )** — zmienne **e, p, d** otrzymują konkretne wartości, np.:

```
e = karczochy-Melania  p = wołowina-z-rusztu  d = lody-gruszkowe
```

Wobec tego, gdy przejdziemy do drugiej części koniunkcji — **ryba(p)** z wartością, jaką otrzymała zmienna **p**, sprawdzać będziemy czy zachodzi: **ryba(wołowina-z-rusztu).** Bank nie zawiera takiego twierdzenia, zatem proponowane wartości **e, p, d** nie spełniają naszego pytania, są więc odrzucane i system próbuje znaleźć inne rozwiązania. Ostatecznie program poda osiemnaście możliwych rozwiązań:

```
e = karczochy-Melania  p = okoń-w-algach  d = lody  gruszkowe

.....

e = jajka-sadzone-z-rukwią  p = sandacz-faszerowany  d = melon-niespodzianka
```

W tym miejscu warto zwrócić uwagę na następujące fakty:

- Aby sprawdzić koniunkcję relacji, sprawdza się ich spełnialność kolejno od lewej strony do prawej.
- Podczas wykonywania obliczeń pewne zmienne mogą otrzymać wartości i wówczas wszystkie wystąpienia tej zmiennej mają tę samą wartość.
- W relacjach nie rozróżnia się argumentów wejściowych i wyjściowych. Również ta sama relacja może występować w wielu rolach: jeśli wszystkie jej argumenty są znane, sprawdza się tylko, czy jest ona spełniona; jeśli pewne ar-

gumenty są nieznane, oblicza się zbiory wartości, jakie można nadać tym argumentom, aby relacja była spełniona.

• Wykonanie jest niedeterministyczne w tym sensie, że wyliczane są wszystkie możliwe wartości argumentów spełniające relację. Dodajmy jeszcze uwagę krytyczną:

• Wbrew wrażeniu, które można odnieść z powyższych rozważań, w PROLOGU nie programuje się zupełnie beztrosko. Istotny wpływ na efektywność i poprawność programów niejednokrotnie ma kolejność klauzul i wywołań (por. [3], s. 43).

Po tych uwagach uzupełnijmy naszą wiedzę o spożywaniu posiłków i wprowadźmy wartość kaloryczną każdego proponowanego dania.

```
"wartość kaloryczna jednej porcji"   (iv)
kalorie(karczochy-Melania, 150) → ;
kalorie(jajka-sadzone-z-rukwią, 202) → ;
kalorie(trufle, 212) → ;
kalorie(wołowina-z-rusztu, 532) → ;
kalorie(kurczak-w-lipie, 400) → ;
kalorie(okoń-w-algach, 212) → ;
kalorie(sandacz-faszerowany, 254) → ;
kalorie(lody-gruszkowe, 223) → ;
kalorie(truskawki-z-Chantilly, 289) → ;
kalorie(melon-niespodzianka, 122) → ;
```

Twierdzenie: **kalorie(sandacz-faszerowany, 254)→;** interpretuje się jako: „podawana porcja sandacza faszerowanego zawiera 254 kalorie". Teraz, aby dowiedzieć się o wartość kaloryczną przystawek, zapytamy się:

```
przystawka(e)  kalorie(e,c);
```

Dla każdej wartości parametru **e,** spełniającej relację **przystawka( )**, pragram nada zmiennej **c** wartość, która spełnia relację **kalorie(e,)**. Uzyskamy zatem odpowiedzi:

```
e = karczochy-Melania          c = 150
e = trufle                     c = 212
e = jajka-sadzone-z-rukwią     c = 202
```

Bardziej ostrożny konsument będzie chciał znać wartość kaloryczną całego posiłku, zdefiniujmy więc relację:

```
wartość(e, p, d, v) → kalorie(e, x)  kalorie(p, y)  kalorie(d, z)
                                        dodaj(x, y, z, v);   (v)
```

gdzie relacja **dodaj(x, y, z, v)** jest spełniona, jeśli **v** jest sumą wartości kalorycznych **x, y, z** składników posiłku. Aby poznać wartości kaloryczne wszystkich posiłków, zadajemy pytanie:

```
posiłek(e, p, d)  wartość(e, p, d, v);
```

i otrzymujemy odpowiedzi:

```
e = karczochy-Melania   p = wołowina-z-rusztu   d = lody-gruszkowe
                                  v = 905
   .....
   .....
e = jajka-sadzone-z-rukwią   p = sandacz-faszerowany
                              d = melon-niespodzianka   v = 578
```

Jako całkiem naturalną kontynuację definiujemy posiłek rozsądny — jako posiłek, którego wartość kaloryczna jest mniejsza od 800 kalorii:

```
posiłek-rozsądny(e, p, d) → posiłek(e, p, d)  wartość(e, p, d, v)
                                  mniejsze(v, 800);   (vi)
```

Pytanie:

```
posiłek-rozsądny(e, p, d);
```

daje w odpowiedzi:

```
e = karczochy-Melania   p = kurczak-w-lipie   d = lody-gruszkowe
                                  .....
e = jajka-sadzone-z-rukwią   p = sandacz-faszerowany   d = melon-
                                                       niespodzianka
```

Zakończmy ten przykład pytaniem:

```
posiłek-rozsądny(e, p, d)  mięso(p);
```

które pozwala na wybór rozsądnego posiłku mięsnego i które daje następujące odpowiedzi:

```
e = karczochy-Melania   p = kurczak-w-lipie   d = lody-gruszkowe
e = karczochy-Melania   p = kurczak-w-lipie   d = melon-niespodzianka
e = trufle              p = kurczak-w-lipie   d = melon-niespodzianka
e = jajka-sadzone-z-rukwią   p = kurczak-w-lipie   d = melon   niespo-
                                                          dzianka
```

Wyklucza to wszelkie posiłki z wołowiną!

Jeśli zbierzemy teraz reguły **(i)** — **(vi)** i dołączymy definicje:

```
dodaj(a, b, c, d) → val(add(a, add(b, c)), d) → ;
mniejsze(x, y) → val(inf(x, y), 1) → ;
```

to otrzymamy pełny PROLOGOWY program karty dań.

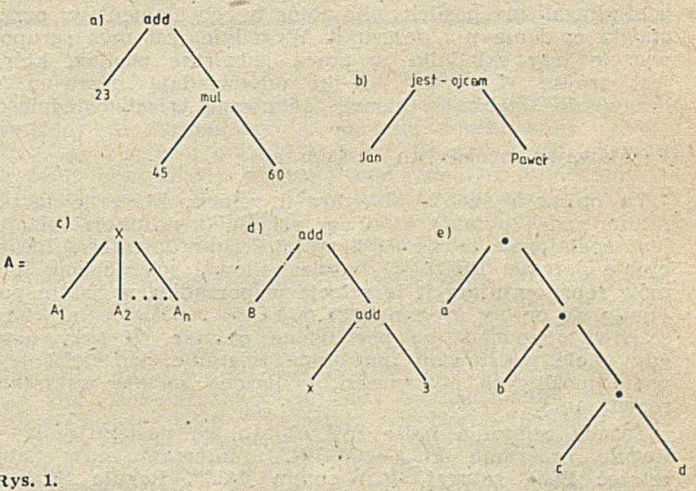
Powyższy przykład umożliwił przedstawienie pewnych cech charakterystycznych PROLOGU:

— definicji relacji pomiędzy obiektami
— pytań dotyczących tych relacji
— zapisu zmiennych
— reprezentacji koniunkcji i alternatywy relacji
— niedeterminizmu wykonania
— braku rozróżnienia argumentów danych i wynikowych

Wróćmy teraz do tych cech, przyglądając się im nieco dokładniej.

Drzewa

W naszym przykładzie jedynymi obiektami, którymi operowaliśmy, były stałe reprezentowane przez nazwę lub wartość. Natomiast zmienne służyły nam do oznaczania obiektów jeszcze nie znanych. W istocie, bardziej ogólną strukturą są drzewa. Np. wyrażenie arytmetyczne **23 + + 45*60** reprezentuje drzewo z rysunku 1a. Relację **jest-ojcem(Jan,Paweł)** przedstawić można za pomocą drzewa z rysunku 1b.



Rys. 1.

Ogólniej — drzewo **A** składa się z węzła **X** zwanego korzeniem i z uporządkowanego zbioru (być może pustego) drzew $A_1,...,A_n$ (por. rys. 1c). $A_1,...,A_n$ są poddrzewami **A** i każde poddrzewo drzewa A_i $(1 \leqslant i \leqslant n)$ jest również poddrzewem **A**. Korzenie $A_1,...,A_n$ są synami **X**. Każdy węzeł bez synów nazywa się liściem.

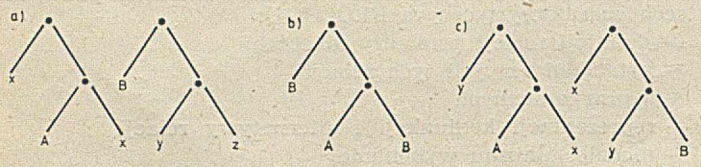
W PROLOGU drzewo może być częściowo nieznane. W tym przypadku jeden z jego liści redukuje się do zmiennej. Np. drzewo z rysunku 1d reprezentuje nieskończoną rodzinę drzew, otrzymanych przez zastąpienie zmiennej **x** dowolnym drzewem. Drzewo bez zmiennych zredukować można do korzenia; stowarzyszonym obiektem jest wówczas stała: identyfikator, łańcuch znaków lub liczba.

Często wygodne jest stosowanie liniowego zapisu drzew. Np. drzewo z rysunku 1d zapisać można jako **add(8,add(x, 3))**, lub w inne notacji — **<add, 8, <add, x, 3 >>**.

Pośród innych struktur danych, ważną rolę odgrywa lista. Listy można tworzyć dzięki infiksowemu operatorowi „.". Np. **a.b.c.d** oznacza listę **(a,b,c,d)**. Zauważmy, że listy można również reprezentować za pomocą drzew. Listę **a.b.c.d** opisuje drzewo z rysunku 1e.

Unifikacja

Podstawową operacją wykonywaną na drzewach jest unifikacja. Jeśli dane są dwa drzewa, zawierające ewentualnie części zmienne, to unifikacja polega na znalezieniu, o ile istnieją, takich wartości, które powinny być przyjęte przez zmienne występujące w jednym lub drugim drzewie, aby oba drzewa były zgodne. Zbiór równości **{zmienna = wartość}**, dla których dwa drzewa są równe, nazywamy podstawieniem. Na przykład — drzewa z rysunku 2a unifikują się w drzewo z rysunku 2b z podstawieniem **{x = B, y = A, z = B}**. Drzew z rysunku 2c nie można zunifikować, gdyż trzeba by było dokonać podstawień **{x = y, x = B, y = A}**.



Rys. 2.

Struktura programu

Program w PROLOGU składa się z ciągu reguł. Każda z nich składa się z głowy (ang. head) i — być może pustego — ogona (ang. tail) reguły, poprzedzonego strzałką. Głowa reguły — to pojedynczy literał, zaś jej ogon — to ciąg literałów. Reguły, literały, argumenty są w PROLOGU termami, przy czym term może być stałą (identyfikatorem, liczbą, ciągiem znaków), zmienną lub termem złożonym (**n**-tką).

Notacja **n**-tek jest użyteczna, gdy chce się zgrupować zbiór wielu termów, aby stworzyć z nich nowy term, którym można potem wygodniej operować. Przykładowo — zdefiniowaliśmy posiłek jako zbiór trzech elementów: przystawki **e**, dania **p** i deseru **d**. Wygodnie jest móc zgrupować te trzy składniki w nową jednostkę **posiłek**, który jest trójką $<e, p, d>$. **N**-tki odpowiadają drzewom i, oczywiście, unifikacja bierze pod uwagę tę odpowiedniość.

Podstawowy mechanizm — ścieranie

Po opisaniu świata obiektów i relacji, które je łączą, można zadać pytanie o to, czy relacja (koniunkcja relacji) jest spełniona dla pewnych wartości argumentów. Intuicyjnie patrząc: PROLOG będzie usiłował ścierać ciąg termów reprezentujących te relacje w porządku, w jakim pojawiły się one w programie, i dojść do podstawienia, które stanowi odpowiedź na postawione pytanie. W przypadku gdy wiele podstawień umożliwia ścieranie, tzn. jeśli jest wiele możliwych odpowiedzi, wyliczone zostają wszystkie rozwiązania.

Zasada ścierania może być objaśniona następująco: Regułę programu **P(...) → Q(...)R(...)**: interpretujemy: „aby zetrzeć **P(...)**, **zetrzyj Q(...)** potem **R(...)**". Reguła **S(...) →**; oznacza: „**S** ściera się". Pytanie: $S_1(...)\ S_2(...)\ ...S_n(...)$; oznacza: „zetrzyj $S_1(...)$, potem $S_2(...)$, ..., potem $S_n(...)$. Np. jeśli reguła $S_1(...) \rightarrow Q_1(...)\ ...\ Q_p(...)$; została wybrana do starcia $S_1(...)$ w powyższym pytaniu, naszym nowym celem jest starcie $Q_1(...) ... Q_p(...)\ S_2(...) ... S_n(...)$. Proces ten kontynuujemy aż do momentu, gdy:

— wszystko jest starte; oznacza to sukces i rozwiązaniem jest podstawienie, które pozwoliło na starcie
— osiągnie się term, którego nie możemy zetrzeć; oznacza to porażkę.

W obu przypadkach nie zatrzymujemy się: przypomnijmy sobie, że w celu starcia term zaczęliśmy od wyboru pierwszej reguły, która pozwalała to uczynić. Inne możliwości pozostały być może otwarte i w trakcie posuwania się zachowaliśmy je w odwodzie. Mamy zatem do czynienia z nawrotami, w których powraca się do ostatniego rozgałęzienia i próbuje ścierać rozpatrywany term w inny sposób, usiłując znaleźć inną odpowiedź na nasze pytanie. Postępuje się tak dopóty, dopóki pozostają niesprawdzone możliwości. Należy zauważyć, że podczas nowej próby starcia termu **L(...)** przestają obowiązywać wszystkie podstawienia zmiennych, które miały miejsce pomiędzy poprzednim wyborem dla **L(...)** i obecnym.

Dla zilustrowania tego mechanizmu wróćmy do naszego początkowego przykładu:

(1) mięso(wołowina-z-rusztu) → ;
(2) mięso(kurczak-w-lipie) → ;
(3) ryba(okoń-w-algach) → ;
(4) ryba(sandacz-faszerowany) → ;
(5) danie(p) → mięso(p);
(6) danie(p) → ryba(p);

i zadajmy pytanie: **danie(p) dif(p, wołowina-z-rusztu)**, gdzie **dif** jest termem, który się ściera tylko wtedy, gdy jego argumenty są różne. Przedstawimy proces ścierania za pomocą drzewa, w którym:

— w węźle występuje bieżący zbiór termów do starcia (cele) i bieżące podstawienie
— gałąź oznacza regułę wybraną do starcia pierwszego termu
— następnikami węzła są nowe zbiory celów, mogące wynikać ze ścierania pierwszego z celów związanych z rozpatrywanym węzłem. Przykład takiego drzewa przedstawia rysunek 3.

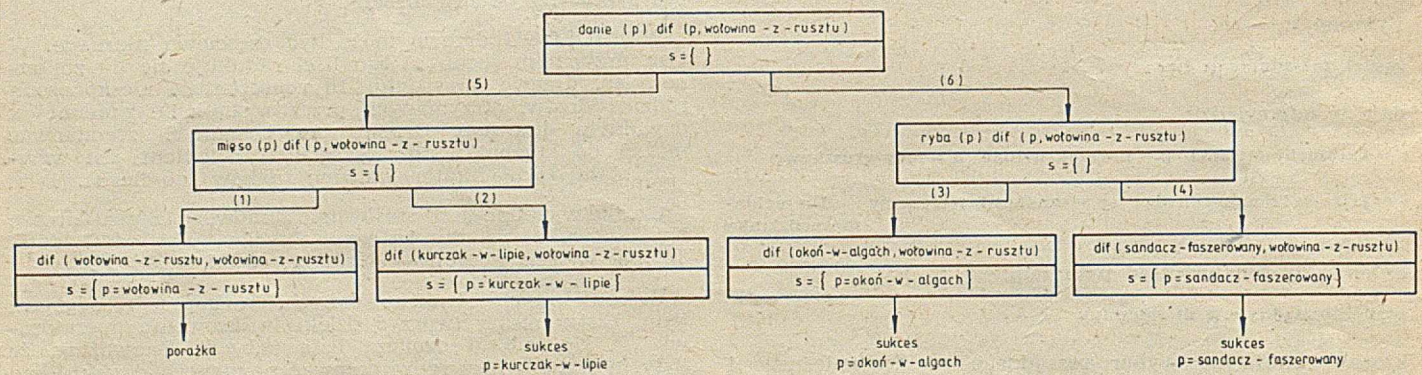
Reguły standardowe

Pewne reguły są znane „z urzędu" przez system, zamiast być definiowane przez użytkownika; nazywa się je regułami standardowymi. Dają one możliwości, które na ogół nie mogłyby być dostępne w czystym PROLOGU. W rzeczywistości to właśnie reguły standardowe czynią z PROLOGU użytkowy język programowania. Mają one często efekty uboczne, zmieniające stan systemu i służą głównie do:

— sterowania wykonaniem (przez modyfikację praw rządzących ścieraniem, niedeterminizmem itd.)
— obsługi wejścia i wyjścia
— sprawdzania typu obiektów
— obliczeń arytmetycznych i obsługi zbiorów
— tworzenia współprogramów.

Operator odcięcia „/"

Pokazaliśmy, że ścieranie termu dokonuje się w sposób niedeterministyczny; system zachowuje w rezerwie różne punkty wyboru, by później do nich powrócić. Wprowadzenie operatora odcięcia do reguły pozwala na usunięcie niektórych punktów wyboru i ewentualnie na otrzymanie



Rys. 3. Drzewo ścierania

programu całkowicie deterministycznego. Zasada wykonania „/" może być wyrażona następująco: starcie „/" daje w efekcie usunięcie wyrobów pozostawionych w odwodzie dla wszystkich termów oczekujących na wytarcie, począwszy od tego, który uruchomił regułę zawierającą „/", aż do tego, który poprzedza „/" w ogonie tej reguły.

Przykładowo jeśli „/" występuje w pytaniu:

```
danie(p) dif(p, wołowina-z-rusztu) / ;
```

to wykonanie „/" odcina możliwość powrotu do relacji **danie** i **dif**. Prześledźmy dokładniej ten przypadek: aby zetrzeć nasze pytanie, zaczynamy (podobnie jak na rysunku 3) od starcia relacji **danie(p)**, uzyskujemy zatem kolejny term do starcia: **mięso(p) dif(p, wołowina-z rusztu) / ;** i dalej: **dif(wołowina-z-rusztu, wołowina-z-rusztu) /.** Term **dif** nie daje się zetrzeć, bowiem argumenty są równe, następuje więc nawrót i próbujemy zetrzeć term **dif(kurczak-w-lipie, wołowina-z-rusztu)** . Relacja **dif** ściera się i wykonujemy operator „/", zatem usuwamy pozostające w odwodzie wybory dla **dif** (w tym przypadku jest to zbiór pusty) oraz wybory dla **danie(p)**, zatem jedyną odpowiedzią będzie tu: **p = kurczak-w-lipie** i PROLOG kończy obliczenia.

Gdybyśmy zadali pytanie:

```
danie(p) / dif (p, wołowina-z-rusztu);
```

sytuacja byłaby nieco inna: w pierwszym kroku uzyskalibyśmy do starcia term **mięso(p) / dif(p, wołowina-z-rusztu),** następnie **dif(wołowina-z-rusztu, wołowina-z-rusztu).** Starcie operatora odcinania powoduje usunięcie możliwości dokonania alternatywnych wyborów dla **mięso(p)** i **danie(p).** Teraz pozostała do starcia jedynie relacja **dif.** Ponieważ argumenty jej są równe, następuje porażka. PROLOG nie szuka innych rozwiązań, bowiem pozostające w odwodzie możliwości zostały odcięte.

JÓZEF B. LEWOC
Wrocław

LITERATURA

[1] Colmerauer A., Les systémes—q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. Raport 43, Dept, d'Informatique, Université de Montréal, 1970

[2] Colmerauer A., Kanoui H., Van Caneghem M.: PROLOG, bases théoriques et developpments actuels. Technique et science informatiques, vol. 2, nr 4, 1983, 271—311

[3] Kluźniak F., Szpakowicz S.: PROLOG. WNT, Warszawa, 1983

[4] Kowalski R., Van Emden M.: The semantics of predicate logic as programming language. JACM, vol. 23, nr 4, 1976, 733—743

[5] Robinson J. A.: A machine—oriented logic based on the resolution principle. JACM, vol. 12, nr 1, 1965, 227—234.

Quo vadis, Jednolity Systemie?

Uwagi, jakie się najczęściej słyszy na temat jakości pracy komputerów Jednolitego Systemu (JS) są, oględnie mówiąc, niezgodne z wypowiedziami, iż system ten stanowi przykład słusznego i dobrze zrealizowanego podejścia do rozwiązania problemu komputeryzacji w Polsce i krajach RWPG. Z drugiej strony — jakość wykonania polskiego sprzętu informatycznego oceniana jest wysoko, o czym świadczy choćby wielkość jego eksportu.

Wymowny przykład

W ramach realizacji Międzyuczelnianej Sieci Komputerowej (MSK) przeprowadzono doświadczenie porównujące jakość pracy o podobnym charakterze pod nadzorem systemów TSO + TCAM na komputerze R-32 oraz MINIMOP na komputerze ODRA 1325. TSO wybrano dlatego, że spośród różnych systemów wielodostępnych zapewnia największe udogodnienia dla użytkownika przy terminalu JS, natomiast wybór TCAM jest uzasadniony tym, że w wersji standardowej stanowi on logiczne uzupełnienie TSO.

Inne standardowe systemy sterujące teleprzetwarzania [4] i metody dostępu [14] nie zapewniają tak dużych udogodnień użytkowych lub nie mogą być użyte na komputerze R-32 (VTAM). MINIMOP był jedynym eksploatowanym w Politechnice Wrocławskiej systemem o podobnym zastosowaniu (w studenckich laboratoriach wielodostępu).

Doświadczenie przeprowadzono dla czterech sesji studenckich; dwóch pod nadzorem TSO (uruchamianie programów w PASCALU i FORTRANIE) oraz dwóch z wykorzystaniem MINIMOPU (uruchamianie programów w FORTRANIE). Wyniki doświadczeń omówiono w [2]. Wyniki te okazały się zaskakujące: koszty wykonania podobnych prac na R-32 są kilkadziesiąt razy wyższe niż na ODRZE 1325. Czas odpowiedzi R-32 ma rozkład zbliżony do wykładniczego z wysoką wartością oczekiwaną (ok. 30 s) i bardzo dużą wartością maksymalną (zmierzona — 435 s, teoretyczna — wiele godzin) przy stosunkowo dobrej obsłudze „głupich" (wymagających małego czasu procesora) i szybkich rozkazów.

Rozkład czasu odpowiedzi przy systemie MINIMOP jest zbliżony do normalnego o wartości oczekiwanej ok. 2 s (poniżej 1 s, jeśli odrzuci się czas załączania silnika dalekopisu) oraz małej wartości maksymalnej (zmierzona — 7 s, teoretyczna — ok. 30 s) przy ośmiu aktywnych terminalach (siedem terminali w przypadku TSO). Parametry były dobrane według zaleceń producenta [5, 15].

Przykład ten potwierdza przypuszczenie, że oprogramowanie systemowe R-32 nie daje użytkownikom pożądanego efektu. W podanym przykładzie główną winę ponoszą TCAM oraz Driver TSO. Ten ostatni jest opisany źle w [5], choć gdzie indziej jest bardzo dobrze udokumentowany [10]. Dokumentacja [10] jest jednak trudno dostępna, niewiele więc osób przeczytało uczciwą wypowiedź autorów na str. 25: „Dobrowolnie przyjęte oddzielenie Drivera (od reszty TSO) ma na celu ułatwienie jego wymiany lub modyfikacje". Wypowiedź ta świadczy o ich wątpliwościach co do jakości działania tego produktu. System TSO, choć atrakcyjny użytkowo, jak wiadomo — nie przyjął się powszechnie.

Warto więc dokładniej przyjrzeć się podstawowej przyczynie tej złej sytuacji: systemowi operacyjnemu OS/MVT [8], dla którego TSO i TCAM stanowią tylko nadbudówki.

Trochę historii

Podczas Szkoły Zimowej nt. sieci komputerowych, zorganizowanej przez Zakład Informatyki Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej, poznałem przedstawiciela Computer Laboratory (CL) Uniwersytetu w Cambridge, który okazał się jednym z autorów procesora komunikacyjnego w systemie PHOENIX [7]. Przedstawiciel ten udzielił konkretnej pomocy, którą wykorzystano przy budowie podsieci komunikacyjnej MSK.

Rok później na zaproszenie Centrum Obliczeniowego Politechniki Wrocławskiej (CO PWr) przyjechało do Polski dwóch dalszych przedstawicieli CL, którzy przywieźli dokumentację źródłową systemu PARROT [19], obejmującą oprogramowanie procesora czołowego oraz zmiany systemu operacyjnego OS/MVT zastępujące TCAM. Zezwolili

oni na kontrolowane przez CL bezpłatne rozpowszechnianie tego oprogramowania oraz przeszkolili ok. dziesięć osób w zakresie systemu PHOENIX, w tym — zmian wprowadzonych do systemu operacyjnego OS/MVT. Ponadto CO PWr otrzymało serię Raportów Technicznych CL (bez praw rozpowszechniania). Materiały te oraz własne przemyślenia wynikłe z lektury dokumentacji JS i doświadczeń praktycznych stanowią podstawę do poniższej oceny systemu operacyjnego OS/MVT.

Ocena systemu operacyjnego OS/MVT

Znana jest firma [2], która dobrze prosperuje sprzedając usługi w zakresie strojenia paramerów systemu operacyjnego OS/MVT. Strojenie takie zwiększa o ok. 50% przepustowość systemów IBM 360. Fakty te najlepiej świadczą o jakości i czytelności dokumentacji dostarczonej przez producenta. Nie warto więc podkreślać, że jest to bardzo duży system operacyjny. Im większy, bowiem, tym lepiej powinien być udokumentowany, a użytkownik powinien mieć prosty zbiór reguł walki z takim molochem.

Jednak samo strojenie parametrów nie daje radykalnej poprawy jakości systemu — podany na wstępie przykład wskazuje bowiem na znacznie większe marnotrawstwo możliwości sprzętu.

Gospodarka pamięcią dyskową

Bardzo duże marnotrawstwo tkwi w źle rozwiązanej gospodarce pamięcią dyskową [9, 18], uwidoczniające się szczególnie w systemach wielodostępnych. Każdy wolumin dyskowy jest opisany w obszarze dyskowym VTOC złożonym z co najmniej dwóch bloków. Dostęp do dowolnego zbioru wymaga znalezienia bloku pierwszego **F1** zawierającego opis zbioru maksymalnie trzech rozłącznych obszarów dyskowych, natomiast trzynaście dalszych może być opisanych w drugim bloku **F3**.

Wolna przestrzeń dyskowa jest opisana osobnym łańcuchem bloków VTOC **F5**. Sam VTOC jest opisany przez osobny blok **F4**.

Programy gospodarki pamięcią (tworzenie, rozszerzanie, skracanie i kasowanie zbiorów) są osobne, mimo że wiele ich funkcji jest identycznych. Są one nakładkowane. Zawsze czyta się i zapisuje ponownie **F4**, wszystkie **F5**, **F1**, **F3**, z tym że przy tworzeniu następuje obieg całego VTOC — celem sprawdzenia, czy nie powtarzają się nazwy.

W typowej sytuacji prowadzi to do ok. dwudziestu transmisji dyskowych na jedną transakcję gospodarki pamięcią. Oprócz tego, pozorne udogodnienia systemu dyskowego, takie jak subalokacja czy organizacja indeksowo-sekwencyjna, prowadzą w systemie wieloprogramowym do znacznie dłuższych czasów dostępu do zbiorów niż rzeczywiście jest to potrzebne. We wspomnianym już CL wymieniono programy gospodarki pamięcią dyskową, zachowując ich interfejsy zewnętrzne (w tym struktury danych). Przestrzeń dyskowa opisano w postaci tablicy bitowej w pamięci operacyjnej. Wyeliminowano sprawdzanie powtarzania się nazw, dzięki wprowadzeniu innych zabezpieczeń. Kod programu jest zwarty i rezyduje w pamięci operacyjnej. Po awarii systemu odtwarza się tablice opisu przestrzeni dyskowych przez pełny przegląd VTOC (w czasie ok. 1 s). Wyeliminowano również niszczenie zbiorów dyskowych podczas awarii systemu.

W wyniku powyższych zmian nastąpiło ok. dziewięciokrotne skrócenie czasu działnia procesora oraz dziewietnastokrotne czasu wykonania typowych transakcji **Allocate** i **Scratch**.

Przydzielanie zasobów i szeregowanie zadań

OS/MVT + TSO nie narzucają żadnych ograniczeń na pobieranie zasobów przez zadania lub sesję użytkownika pracującego w podziale czasu. Pozornie — bo np. wprowadzanie OS MVT, TSO + TCAM oraz trzech regionów po 110 K słów pozostawia na R-32 — o maksymalnej pamięci 1 MB dostarczanej przez producenta — zaledwie 32 KB na zadania wsadowe. W obszarze tym nie mieszcza się nawet wieksze kompilatory, a o efektywnym wykorzystaniu procesora dzięki zastosowaniu wieloprogramowości nie ma nawet co marzyć.

We wspomnianym doświadczeniu procesor wykorzystywany był dla realizacji zadań drugoplanowych nie więcej niż w 2%, mimo przygotowania pełnego i odpowiedniego wsadu na cały okres próby, i tylko w ok. 10% — dla zadań pierwszoplanowych (sesje).

Brak ograniczeń może wprawdzie cieszyć użytkownika, mniej go jednak cieszą wynikające stąd koszty przetwarzania, wynoszące 5—10 tys. zł za jedną godzinę czasu pracy zestawu. W CL narzucono ograniczenia na wielkość sesji: normalny region wymian ma pojemność 34 KB, natomiast programy dłuższe mogą działać w drugim regionie bez wymian. System szeregowania zadań oparty jest na zasadzie przydzielanej dla projektu tzw. ceny jednostkowej, która maleje wraz ze wzrostem łącznego rzeczywistego wykorzystania zasobów przez projekt. W wyniku tego założenia użytkownicy zmuszeni są do oszczędnego wykorzystania maszyny, co przynosi im oczywiste korzyści.

Szeregowanie zadań w OS/MVT z systemem HASP odbywa się na podstawie łącznej ilości przeznaczonych do wykonania prac. Powoduje to, że rozkład czasu do wykonania zadania będzie miał charakter wykładniczy (jak dla sesji w doświadczeniu). Nigdy więc nie wiadomo, czy i kiedy wykona się poszczególne zadania. W CL wszystkie możliwe zadania kwalifikuje się do natychmiastowego wykonania (zwykle jest to ponad 85% zadań [7]), natomiast pozostałe otrzymują czas wykonania trzy godziny lub według życzenia użytkownika (np. w nocy). Użytkownik jest uprzedzony o długim czasie wykonania i może zrezygnować z pracy w danych warunkach.

W OS/MVT brak jest mechanizmów zachęcających do oszczędnego wykorzystania pamięci dyskowych. W wyniku tego wielu użytkowników marnuje cenne zasoby, ograniczając dodatkowo możliwości wykorzystania całego systemu (konieczność częstych zmian dysków). Do takiego postępowania zmusza zresztą brak dobrej protekcji zbiorów. W CL zastosowano wspólne obszary zbiorów, zabezpieczenie przed nieupoważnionym zapisem oraz wspomniane już rozliczanie, zachęcające do oszczędnego wykorzystywania pamięci dyskowej.

Przydział zasobów zastosowany w CL opiera się na rozpatrywaniu poszczególnych użytkowników, a nie wydziałów, co pozwala również uzyskać znaczne oszczędności.

Szkoda, że w wielu poważnych opracowaniach nt. zastosowań teorii kolejek do badań systemów komputerowych (np. [11]) zapomina się o elementarnych zasadach gospodarności błyskotliwych nieraz rozważań na temat momentów rozkładów szukanych zmiennych losowych.

Niezrozumiałe jest dla mnie powszechne pomijanie przykładów podanych w podstawowej monografii z rachunku prawdopodobieństwa [6], które wskazują bardzo wysokie wartości wariancji w porównaniu z wartościami oczekiwanymi. OS/MVT + TSO potwierdza tę prawidłowość.

Fragmentacja pamięci

W rozwiązaniu OS/MVT dowolne zadanie może zostać załadowane w dowolne pole pamięci operacyjnej, natomiast musi być ono wykonywane tylko w tym polu, do którego zostało załadowane. W wyniku tego następuje fragmentacja pamięci, o której pisze się poważne rozprawy [13], a która — moim zdaniem — jest wynikiem prostego błędu projektu oprogramowania i (lub) sprzętu — złego wykorzystania udziwnionego mechanizmu protekcji pamięci przy użyciu tzw. kluczy ochrony, których nie zmienia się dynamicznie.

W CL częściowo zmniejsza się straty płynące z fragmentacji pamięci, wykorzystując jako pola do składowania sesji — „dziury" powstające między polami zadań drugoplanowych [16]. W ten sposób uzyskano pewien przyrost liczby jednocześnie wykonywanych sesji. Ale CL nie musiało rozwiązywać tego problemu od podstaw: zwiększono tam pamięć operacyjną z 1 MB do 4 MB, a znaczna część obciążenia systemu stanowią zadania krótkie (edycja i uruchamianie krótkich programów przez studentów). W naszym kraju zadania drugoplanowe stanowią podstawę obciążenia maszyn JS, dlatego też wyeliminowanie fragmentacji pamięci jest tym bardziej celowe.

Zawodność oprogramowania

OS/MVT zawiera błędy prowadzące do awarii systemu i konieczności ponownego ładowania, przy czym niekiedy niszczone są również zbiory dyskowe. W CO PWr obserwuje się tzw. upadki (załamania) systemu z częstością większą niż raz dziennie. CL nie rozwiązało tego problemu, lecz ograniczyło skutki upadków. Wspomniane wcześniej zmiany programów gospodarki pamięcią dyskową wyelimnowały

niszczenie zbiorów dyskowych, natomiast straty powodowane ładowaniem systemu zmniejszono przez zastosowanie szybkiego standardowego programu ładującego (Fast IPL).

Przy ponad stu instalacjach R-32 w Polsce należy zabrać się do zlokalizowania tych błędów co — moim zdaniem — okaże się nietrudne po dokładnym poznaniu wnętrza OS/MVS (przynajmniej w podejrzanych okolicach!). Można też zbudować sprzętowo-programowy system automatycznej odnowy systemu, ograniczający do minimum skutki przemijających zakłóceń sprzętowych i programowych. Jak wykazuje praktyka, dokładna znajomość systemu operacyjnego nie jest wówczas niezbędna [3].

Wymienione powyżej wady systemu operacyjnego OS/MVT wyjaśniają już w części przyczyny negatywnych opinii o maszynach JS. W teorii czy dokumentacji wszystko jest w porządku, a dopiero praktyka wykazuje nieprawdziwość lub istotną niepełność opisów słownych. Ale wówczas jest już za późno: jest system, bogate oprogramowanie i trzeba je eksploatować.

W Polsce eksploatowanych jest ok. 120 egzemplarzy R-32 w różnych zestawach, w tym informatyczne giganty na kaczych łapach, jak to popularnie określa się konfiguracje z pamięcią 12, a nawet 256 KB. Zamrożone w nich nakłady inwestycyjne można oszacować jako 10^{10} zł, a efekty są niższe niż uzyskiwane przy wielokrotnie niższych nakładach na taką samą liczbę komputerów rodziny ODRA 1300.

Historii ciąg dalszy

Posiadając materiały z CL oraz możliwość uzyskania pełnych programów źródłowych, próbowałem w kilkuosobowym, nieformalnym zespole zainicjować wdrożenie modyfikacji OS/MVT + TSO + TCAM w trzech instytucjach:
— uczelnianym ośrodku obliczeniowym
— dużym ogólnodostępnym ośrodku obliczeniowym
— u dostawcy systemów komputerowych.

Celem tej inicjatywy było przede wszystkim wyszkolenie kadry, zdolnej do wdrażania takich zmian u wielu użytkowników i — co ważniejsze — opracowywanie nowych modyfikacji oprogramowania czy sprzętu JS. W wyniku faktycznego braku zainteresowania tych instytucji zespół po półtorarocznych próbach zrezygnował z wdrażania zmian opracowanych w CL, ograniczając zakres działań do wymiany TCAM na PARROT [1] oraz zmian Drivera TSO, niezbędnych dla skutecznej pracy systemów sieciowych. Przyczyną rezygnacji była niemożliwość uzyskania środków finansowych rzędu kilku mln zł na opłacenie kosztów maszyny i robocizny oraz przywiezienie kompletu oprogramowania.

Brak zainteresowania tą tak istotną, z ogólnospołecznego punktu widzenia, inicjatywą można wyjaśnić następująco:

- Przy braku dostatecznej liczby zleceń, ogólnodostępny ośrodek obliczeniowy rozliczany i oceniany jest według czasu wykorzystania zestawu, a nie ilości rzeczywiście wykonanych prac [20]. Zatem im drożej się obsługuje użytkowników, tym lepiej.
- Dostawca zainteresowany jest sprzedażą nowych systemów, a nie poprawianiem wydajności sytemów wcześniej dostarczanych, gdyż to ogranicza jego rynek zbytu.
- Ośrodek uczelniany żyje własnym życiem, sprowadzając swą działalność również do sprzedaży usług podobnie jak ośrodek ogólnodostępny.

Tymczasem gdzie indziej powstał nowy zespół, któremu przełożeni umożliwili skuteczne wdrażanie zmian OS/MVT. Można żywić nadzieję, że prace te dadzą w bieżącym roku konkretne wyniki. Zespołowi temu należy skutecznie pomagać w jego pracy, bo nie jest ważne, czy praca zostanie wykonana we Wrocławiu, czy gdzie indziej. Przytoczone fakty uzasadniają, że musi być ona wykonana.

Możliwości dalszego postępowania

Sprawa maszyn JS, czy szerzej — przyszłości dużych (na naszą skalę) komputerów w Polsce jest obecnie przedmiotem zainteresowania wielu ludzi i instytucji. Istnieje teoretycznie możliwość podjęcia następujących decyzji:

1) zaprzestać produkcji maszyn JS i rozwijać serię ODRA 1300
2) zaprzestać produkcji maszyn serii ODRA 1300 i rozwijać JS
3) kontynuować produkcję serii ODRA 1300 i poprawiać parametry użytkowe maszyn JS.

Moim zdaniem należy przyjąć wariant 3 (zgodny z opiniami użytkowników, zebranymi przez Sekcję Maszyn i Systemów Cyfrowych SEP we Wrocławiu [17]) i zastanowić się nad sposobem poprawy własności użytkowych maszyn JS. Widać tu trzy następujące drogi postępowania:

1) rozpocząć produkcję nowych modeli JS z pamięcią wirtualną i innymi dodatkowymi udogodnieniami, pozwalającymi na pracę pod nadzorem systemu operacyjnego OS/VS, rezygnując równocześnie ze zmian w oprogramowaniu R-32
2) nie rozpoczynać produkcji nowych modeli JS, a modernizować oprogramowanie systemowe R-32
3) przygotowywać produkcję nowych modeli JS i równocześnie modernizować R-32.

Wariant 1 to rezygnacja z efektywnego wykorzystania wspomnianej olbrzymiej kwoty nakładów zamrożonych w systemach na R-32. Najlepszy jest chyba kompromisowy wariant 3, wymagający jednak bardziej szczegółowego określenia sposobu rozwiązania oraz zakresu modernizacji R-32. Możliwe są tu dwa podejścia:
— opracowanie całego systemu operacyjnego od nowa przy zachowaniu interfejsów zewnętrznych
— kolejne eliminowanie „wąskich gardeł" w OS/MVT.

Podejście drugie jest ostrożne, ale podobnie jak w CL daje bardzo duże efekty przy stosunkowo niewielkich nakładach.

Tak czy inaczej rozmiary zamrożonych nakładów inwestycyjnych oraz potencjalnych efektów społecznych modernizacji istniejącego oprogramowania uzasadniają jak najszybsze podjęcie omawianych prac.

* * *

Przedstawione w artykule problemy można ująć w postaci następujących wniosków:

- system operacyjny OS/MVT (oraz TSO + TCAM) uniemożliwiają efektywne wykorzystanie możliwości sprzętu JS
- OS/MVT należy modyfikować stopniowo, eliminując kolejno jego „wąskie gardła"
- technologia przetwarzania oraz systemy rozliczeń powinny zachęcić użytkowników do oszczędnego wykorzystywania zasobów komputera
- należy zmienić sposób rozliczania i oceny działalności ośrodków obliczeniowych oraz dostawców systemów komputerowych, tak aby zachęcić ich do zwiększania efektywności wykorzystania sprzętu
- należy systematycznie poprawiać parametry użytkowe maszyn JS, nie przerywając jednak produkcji komputerów serii ODRA 1300
- przygotowanie produkcji nowych modeli JS nie zwalnia od intensywnej modernizacji R-32.

LITERATURA

[1] Bernardyn J. i in.: Projekt oprogramowania procesora czołowego mc. R-32 w sieci komputerowej MSK (na EC 8371), MSK-PK-PC-8371-DP/1. Raport CO PWr, SPR nr 29/82, Wrocław, 1982
[2] Bicz T. i in.: Wstępna ocena opóźnień czasowych przy pracy konwersacyjnej w systemach komputerowych R-32 i Odra 1325. Biuletyn MERA (zgłoszono do druku)
[3] Czesnowski R.: Automatyczny restart komputerowego systemu sterowania i kontroli. Prace IASE, Zeszyt 31, Wrocław, 1977
[4] Dutkowski K. i in.: Analiza wybranych systemów sterujących teleprzetwarzaniem. ZOWAR, 1972
[5] ELWRO, OS/JS: TSO — Podręcznik programisty programowego. Dokumentacja fabryczna nr 4.0201.00000-01 34.037
[6] Feller W.: Wstęp do rachunku prawdopodobieństwa. Tom II, WNT. Warszawa 1969
[7] Hazel P.: Resourse Allocation and Job Scheduling. Raport Techniczny nr 13, University of Computer Cambridge Laboratory, 1980
[8] IBM System/360 Operating System, MVT Guide, GC28-6720-4, 1972
[9] IBM System/360 Operating System, OS DADSM Logic Release. 21,7, 94, 28-6607-9, 1974
[10] IBM System/360 Operating System, Time Sharing Option (TSO). Control Program, GY27-7199-3, 1973
[11] Kleinrock L.: Queueing systems. John Wiloy and Sons

[12] Kreuter L. Associates: Computer Performance Measurements and Capacity Planning. Dumont, N. J., 1979
[13] Madnick S. E., Donovan J. J.: Operating Systems. Mc Graw Hill, 1977
[14] Niemczycki L.: Oprogramowanie teleprzetwarzania maszyn Jednolitego Systemu, WNT, Warszawa, 1979
[15] Politechnika Wrocławska: System operacyjny MINIMOP. Biblioteka WASC, Wrocław, 1974
[16] Powers J. S.: Store to Store Swapping for TSO under OS/MVT. Raport Techniczny nr 14, University of Cambridge Computer Laboratory, 1980
[17] SMiSC przy OWr SEP: Pismo do KZNOT przy Centrum MERA-ELWRO dnia 10.03.1981
[18] Stoneley A.J.M.: A Replacement for the OS/360 Dise Space Management Routines. Raport Techniczny nr 3, University of Cambridge Computer Laboratory, 1975
[19] Stoneley A.J.M.: PARROT — A replacement form TCAM. Raport Techniczny nr 5, University of Cambridge Computer Laboratory, 1976
[20] ZETO. Cennik nr 1—U/80, Usługi informatyczne. Warszawa 1980.

PIOTR FUGLEWICZ
CNPSS MERASTER
Katowice

MODULA-2 — język lat osiemdziesiątych (3)

Implementacja ETH

Trzecia część artykułu opisuje implementację systemu MODULA-2 wykonaną w Eidgenossische Technische Hochschule (ETH) w Zurychu przez grupę w składzie: Leo Geissmann, Svend Erik Knudsen, Christian Jacobi, Hans Heinrich Naegeli i Anton Gorrengourt, pod kierownictwem prof. N. Wirtha. System działa na komputerach serii PDP-11, pod nadzorem systemu operacyjnego RT-11, i składa się z kompilatora, programu łączącego, uzdatniającego oraz egzekutora zawierającego program ładujący. Jednostki danych akceptowane przez składniki są plikami systemu RT-11.

EGZEKUTOR

Egzekutor jest programem rezydentnym o nazwie MODULA. Po zainicjowaniu tego programu pod systemem RT-11, jest mu przekazywane sterowanie.

Egzekutor akceptuje nazwy plików programów z domyślnym rozszerzeniem [1]) LOD i — używając swojego programu ładującego — ładuje program użytkowy do pamięci oraz przekazuje mu sterowanie. Po zakończeniu wykonywania programu, sterowanie jest zwracane do egzekutora, który wykazuje gotowość do przyjęcia kolejnej nazwy, wyświetlając na pulpicie operatora gwiazdkę (*). W skład funkcji egzekutora wchodzi m.in. obsługa przerwań wewnętrznych programu, zapewniających współpracę programu z otoczeniem.

[1]) Nazwa pliku w konwencji RT-11 składa się z 12 znaków; pierwsze trzy określają urządzenie, sześć następnych stanowi faktyczną nazwę, a trzy ostatnie — tzw. rozszerzenie (ang. extension), interpretowane jako typ pliku.

Mgr inż. PIOTR FUGLEWICZ ukończył Wydział Automatyki i Informatyki Politechniki Śląskiej w Gliwicach, w roku 1979, broniąc pracę na temat systemów operacyjnych. Początkowo pracował w Instytucie Systemów Sterowania, a od 1982 roku w Centrum Naukowo-Produkcyjnym Systemów Sterowania MERASTER w Katowicach — jako projektant. Zajmuje się oprogramowaniem narzędziowym dla małych systemów komputerowych, teorią języków programowania i przetwarzaniem tekstów.

KOMPILATOR

Kompilator (o nazwie COMP) sam jest programem w języku MODULA-2. Po załadowaniu żąda podania nazwy pliku, który ma być kompilowany. Domyślnym rozszerzeniem nazwy pliku modułu programowego jest MOD, a dla modułów definicyjnych — DEF.

Kompilator tworzy plik wydruku o nazwie takiej samej jak plik wejściowy i rozszerzeniu LST. W przypadku pomyślnego zakończenia kompilacji, tworzony jest również plik kodu wynikowego (ang. object file) o rozszerzeniu LNK oraz — dla modułów definicyjnych — tablica symboli z rozszerzeniem SYM. Dodatkowo tworzony jest plik do ewentualnego wykorzystania przez program uzdatniający z rozszerzeniem REF (ang. reference).

Oprócz dostępu do tekstu źródłowego kompilator musi mieć dostęp do plików tablic symboli (SYM) wszystkich modułów wyspecyfikowanych na liście importowej kompilowanego programu.

Kompilator jest pięcioprzebiegowy. Informacje między kolejnymi przebiegami są przekazywane przy użyciu plików dyskowych. W każdym przebiegu kompilator zapisuje informacje na przemian w jednym z dwóch plików, o długości ok. 30 tys. słów. Trzeci plik, o długości ok. 5000 słów, służy do przechowywania tablicy identyfikatorów kompilowanego programu przez cały czas trwania kompilacji.

W kolejnych przebiegach kompilator realizuje następujące czynności:
— sprawdza poprawność syntaktyczną jednostki kompilacji, odczytuje zbiory symboli importowanych modułów oraz tworzy tablice identyfikatorów
— analizuje deklaracje jednostki kompilacji i tworzy plik odwołań zewnętrznych (ang. cross-references)
— sprawdza zgodność typów w treściach procedur
— generuje kod dla wyrażeń
— generuje kod dla instrukcji i tworzy plik wynikowy **(LNK).**

Kompilacja modułu definicyjnego jest wykonywana tylko w dwóch pierwszych przebiegach. Jeśli wykonuje się pomyślnie — to po drugim przebiegu zapisywana jest tablica symboli i plik wydruku, jeśli nie — tylko plik wydruku. W przypadku wykrycia błędu w pierwszych trzech przebiegach kompilacji modułu programu, kolejne przebiegi nie są wykonywane i na nośnik zapisywany jest tylko plik wydruku z diagnostyką błędów.

SPROSTOWANIE. W poprzednim numerze podaliśmy drugą część artykułu Piotra Fuglewicza — jako trzecią. Przepraszamy.

PROGRAM ŁĄCZĄCY

Po kompilacji, kod wynikowy skompilowanego modułu musi być połączony z kodami importowanych obiektów. Proces łączenia jest wykonywany przez program napisany również w języku MODULA-2.

Wynikiem działania programu łączącego jest plik zawierający postać programu wykonywalną (ang. executable) pod systemem MODULA-2. Plik programu wykonywalnego ma rozszerzenie **LOD** i jego nazwa staje się nowym poleceniem systemowym, którego działanie polega na wykonaniu tego programu.

Program musi być dołączony do istniejącej bazy. W normalnym trybie pracy bazą jest — liczący ok. 2000 słów — podstawowy program wykonawczy (ang. basic executive), zawierający sprzęg programu z systemem operacyjnym RT-11.

Dla każdej jednostki kompilacji, kompilator tworzy tzw. klucz modułu. Klucz modułu generowany w oparciu o zegar systemowy jest niepowtarzalny i służy do rozróżniania kolejnych skompilowanych wersji tego samego modułu. Klucz jest zapisywany do tablicy symboli, postaci wynikowej i wykonywalnej programu. Niezgodność kluczy łączonych modułów powoduje tzw. błąd wersji sygnalizowany przez program łączący.

W omawianej implementacji nie występuje opisana w pierwszej części artykułu możliwość pracy modułów niepołączonych. Możliwa jest jednak praca programów nakładkowanych — dzięki bibliotecznej procedurze **Call.**

Dodatkowe polecenie systemu MODULA-2 — SYSGEN — umożliwia tworzenie programów wykonywanych wprost pod kontrolą systemu operacyjnego RT-11.

PROGRAM UZDATNIAJĄCY

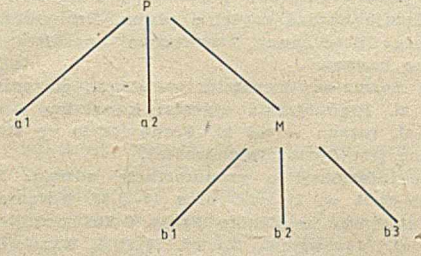
Podobnie jak dwa poprzednie programy program uzdatniający o nazwie DEBUG jest napisany w języku MODULA-2. Jest to program służący nie tyle bezpośredniemu uruchamianiu innych programów: ile raczej badaniu stanu programu, którego działanie zakończyło się błędem.

W chwili wystąpienia błędnej sytuacji system dokonuje zrzutu obrazu pamięci operacyjnej (ang. dump), zapisując jej zawartość na dysk. Program uzdatniający korzysta z pliku, w którym zapisany jest obraz pamięci oraz plików wydruku, tablicy odwołań zewnętrznych i postaci wykonywalnej.

Informacja na temat programu jest dostępna przez cztery tzw. okna (ang. windows):

- Okno **P** jest reprezentacją ciągu procedur aktywnych w chwili wystąpienia błędu. Jest to ciąg obiektów, który zaczyna się od bieżącego procesu (programu), a kończy na obiekcie, który spowodował wystąpienie błędu. Pod pojęciem obiektu rozumie się procedurę lub lokalne zainicjowanie modułu.
- Okno **D** stanowi dostęp do aktualnych wartości zmiennych, należących do inkarnacji (ang. incarnation) procedury lub modułu.
- Okno **T** zapewnia dostęp do tekstu źródłowego modułu
- Okno **C** zapewnia dostęp do dwójkowej postaci informacji w pliku zrzutu pamięci.

Praca z programem uzdatniającym polega na wyborze dowolnego obiektu przez okno **P.** Z kolei, przez inne okna uzyskuje się dostęp do zmiennych, tekstu i postaci dwójkowej tego obiektu.



Struktura drzewiasta odpowiadająca procedurze P

Zmienne procedury wybranej przez okno **P** przedstawiają dla okna **D** strukturę drzewiastą. Każda lokalna zmienna i moduł są węzłami, jak przedstawiono na rysunku odpowiadającym poniższemu przykładowi:

```
PROCEDURE P;
  VAR a1, a2
  ...
  MODULE M;
    VAR b1, b2, b3
    ...
  END M;
  ...
END P;
```

Użytkownik może wybierać interesujące go zmienne, zmieniając poziom, a następnie wybierając zmienną z danego poziomu.

BIBLIOTEKA SYSTEMOWA

W skład biblioteki systemowej wchodzą tzw. standardowe moduły udogodnień (ang. utility). Funkcje realizowane przez te moduły nie stanowią częśei definicji języka. Każda implementacja systemu, a nawet każdy programista — mogą mieć własne rozwiązania sposobu realizacji tych funkcji. Jednakże dla zachowania przenośności oprogramowania zaleca się, aby moduły te były dostępne w każdej implementacji języka.

Wejście-wyjście

Moduł **InOut** eksportuje stałą **StrLeng**, typ **String** oraz procedury: **OpenIO, CloseIO, Read, ReadInt, Write, WriteInt, WriteCard, WriteOct, WriteHex, WriteLn, WriteString, ShowString.**

Operacje wejścia-wyjścia realizowane przez ten moduł umożliwiają wprowadzanie i wyprowadzanie informacji w różnych formach. Poszczególne procedury służą do konwersji ciągów znaków wprowadzanych ze standardowego urządzenia wejściowego, którym jest klawiatura pulpitu operatora, na postać wewnętrzną, oraz — do realizaąji odwrotnych operacji w celu wyprowadzenia informacji na ekran pulpitu.

Procedury **OpenIO** i **CloseIO** umożliwiają pracę z dowolnym plikiem w formacie RT-11.

Strumienie (ang. streams)

Moduł **Streams** eksportuje typ **STREAM** oraz procedury **Connect, Disconnect, Reset, WriteWord, WriteChar, EndWrite, ReadWord, ReadChar, EOS, GetPos, SetPos.**

Strumień jest ciągiem znaków lub słów wysyłanych lub odbieranych przez program i musi być przyłączony do pliku, który powinien być przedtem otwarty. Przyłączony strumień może być odczytywany bądź zapisywany jako ciąg znaków lub słów.

Typ elementów strumienia jest definiowany w chwili przyłączania pliku, a elementy muszą być odczytywane (zapisywane) przy użyciu procedur **ReadChar** lub **Read Word WriteChar** lub **WriteWord.** Każde wywołanie procedury powoduje sekwencyjne odczytanie (zapisanie kolejnego elementu). Zapisywanie strumienia musi być zakończone wywołaniem procedury **EndWrite,** po czym strumień może być odłączony od pliku, a plik — zamknięty.

Pliki

Moduł **Files** eksportuje typ **FILE** oraz procedury **FileName, Lookup, Create, Delete, Release, Close, WriteBlock, ReadBlock, Rename, SetBlock, TransmitBlock, Rad50name, Radix50, Errcode.**

System plików jest określony przez strukturę systemu operacyjnego, pod nadzorem którego działa system MODULA-2. Inne implementacje mogą zbliżać się do istniejących przez utworzenie systemów niezależnego sprzęgu z otoczeniem. Omawianą implementację zaprojektowano dla systemu RT-11, w którym pliki są ciągami bloków, ponumerowanych w kolejności 0, 1, 2 itd., odpowiadających sektorom dyskowym. Każdy blok składa się z 512 bajtów.

Plik jest identyfikowany przez zmienną typu FILE, która w terminologii RT-11 odpowiada numerowi kanału. Numer kanału jest przypisywany plikowi bądź przez wywołanie procedury **Lookup** (dla plików już istniejących na dysku i w skorowidzu) bądź **Create** (przy tworzeniu nowego pliku). Nazwa nowego pliku jest umieszczana w skorowidzu po wywołaniu procedury **Close** (a nie **Create**). Kanał jest zwalniany po użyciu procedury **Close** lub **Release** (jeśli nazwa pliku nie zostanie umieszczona w skorowidzu). Nazwy procedur w tym module odpowiadają w zasadzie nazwom tzw. żądań programowych ang. program request) systemu RT-11 służących do obsługi analogicznych funkcji w samym systemie.

Nakładkowanie

Moduł Loader eksportuje procedurę **Call** oraz zmienną **FirstFree**, która określa adres wolnego obszaru w pamięci. Procedura **Call** umożliwia tworzenie programów nakładkowanych, przez wywołanie w treści programu innych programów w formacie **LOD**, a zmienna **FirstFree** pozwala określić miejsce ładowania nakładki w pamięci.

Obsługa pulpitu

Moduł **TTIO** służy do obsługi terminala i eksportuje procedury **Read, Readagain, Write, SetMode, WriteLn, WriteString.** Transmisja odbywa się znakowo, z możliwością jednokrotnego ponownego odczytania ostatnio wprowadzonego znaku.

Zarządzanie pamięcią

Moduł **Storage** eksportuje procedury **ALLOCATE, DEALLOCATE, SetMode,** które są przeznaczone do przydzielania i zwalniania pamięci przy dynamicznej alokacji zmiennych.

Przy deklaracjach:
TYPE T = ...
VAR w: POINTER TO T;
instrukcje NEW(w) i DISPOSE(w) są tłumaczone przez kompilator na:
ALLOCATE(w, TSIZE(w))
DEALLOCATE(w, TSIZE(w))

Zarządzanie procesami

Moduł **PROCESSSCHEDULER** eksportuje typ **SIGNAL** oraz procedury **STARTPROCESS, SEND, SENDDOWN, WAIT, DOIO, PAUSE, INITSIGNAL,** które w oparciu o mechanizm współprogramów realizują elementarne funkcje związane z pojęciem oprogramowania współbieżnego.

Procedura **STARTPROCESS** aktywizuje proces, a jej parametrami są: desygnator uaktywnianej procedury oraz obszar, w którym powinna pracować.

Typ **SIGNAL** i działające na nim operatory **SEND** i **WAIT** służą do komunikacji między procesami. Wywołanie **SEND(s)** uaktywnia dokładnie jeden proces oczekujący na s, jeśli taki proces istnieje. Sygnały są realizowane przez kolejkę (listę połączoną) deskryptorów procesów. Zmienna sygnałowa zawiera wskaźnik do początku kolejki. Procedura **SIGNAL** odłącza pierwszy element kolejki, a **WAIT** dołącza element na końcu kolejki. Tak więc, przy pobieraniu elementów z kolejki obowiązuje zasada **FIFO.**

Wszystkie procesy są dodatkowo połączone w pierścień który służy do znajdowania następnego procesu po zawieszeniu bieżącego procedurą **WAIT.**

* * *

System opracowany w ETH ma cechy pierwszej implementacji i nie zawiera udogodnień w pełni profesjonalnych. Przykładowo — diagnostyka błędów polega tylko na podawaniu numeru błędu, a programy napisane w MODULI-2 pod tym sytemem nie mogą w prosty sposób korzystać z już istniejących bibliotek systemu RT-11.

Można jednak przypuszczać, że MODULA-2 przejdzie, podobnie jak PASCAL, dość szybko drogę od zastosowań laboratoryjnych do przemysłowych. Już obecnie co najmniej dwie instytucje oferują profesjonalne systemy z MODULĄ-2. Pierwszą z nich jest MODULA Research Institute, który produkuje mikrokomputer LILITH, a drugą — firma VOLITION SYSTEMS, która sprzedaje system MODULA-2, m.in. dla mikrokomputerów APPLE, SAGE oraz innych opartych na mikroprocesorach TI-9900 oraz 8080/Z-80.

Można więc przypuszczać, że MODULA-2 ma szanse stać się PASCALEM lat osiemdziesiątych.

LITERATURA

[1] Geissmann L.: A User Guide to the Modula-2 System. ETH Zurich, 1981

[2] Geissmann L.: Overview of the Modula-2 Compiler. ETH Zurich, 1981

[3] Ramsey D., Gleaves R.: Structured Languages Such as Modula-2 Boost Programmer Output. Computer Technology Review, Summer 1983

[4] Ventura A., Geissmann L.: Overview of the Modula-2 Debugger, ETH Zurich, 1981.

System CAMAC będzie ojcem współczesnych, modularnych systemów cyfrowych — doskonalszych, opracowanych w innej technologii, ale opartych na dotychczasowych, sprawdzonych koncepcjach. CAMAC może stanowić wzór podejścia do normalizacji — np. w stosunku do nieskoordynowanego rozwoju krajowych systemów mikrokomputerowych. Wreszcie — CAMAC dostarcza przykładów, jak budować aparaturę zawierającą komputer, a także — jak włączyć komputer do większego systemu (czego ilustracją są obydwa poniższe artykuły). Warto sobie również uświadomić, że POLON, główny producent tej aparatury w Polsce, jest największą fabryką CAMACA w Europie, jeśli brać pod uwagę liczbę modułów i roczną wartość sprzedaży.

CAMAC na pewno nie zdominuje krajowej informatyki, stanowi jednak dość wyraźne pole na jej mapie. Dwa lata temu poświęciliśmy mu niemal cały numer (4—5, 1982), obecnie — kontynuujemy temat. (Red.)

JACEK MIRKOWSKI
ADAM PIĄTKOWSKI
Zakład Elektroniki Jądrowej i Medycznej
Instytut Radioelektroniki
Politechnika Warszawska

Laboratorium dydaktyczne w systemie CAMAC z rozłożoną inteligencją

Współczesny stan techniki umożliwia skonstruowanie i oprzyrządowanie takiego laboratorium, które pozwala na maksymalne skupienie uwagi eksperymentatora lub studenta na istotnej treści badanego zjawiska, stwarzając mu jednocześnie możliwość dostosowania aparatury oraz metod analizy do rodzaju eksperymentu. Zadania te spełnia lokalna sieć komputerowa wraz z apraturą pomiarową CAMAC.

Poniżej opisano laboratorium oparte na systemie CAMAC przeznaczone do prowadzenia eksperymentów naukowych i dydaktycznych oraz do automatyzacji pracowni studenckich na wydziałach technicznych i fizycznych szkół wyższych.

Laboratorium spełnia dwie podstawowe funkcje dydaktyczne:

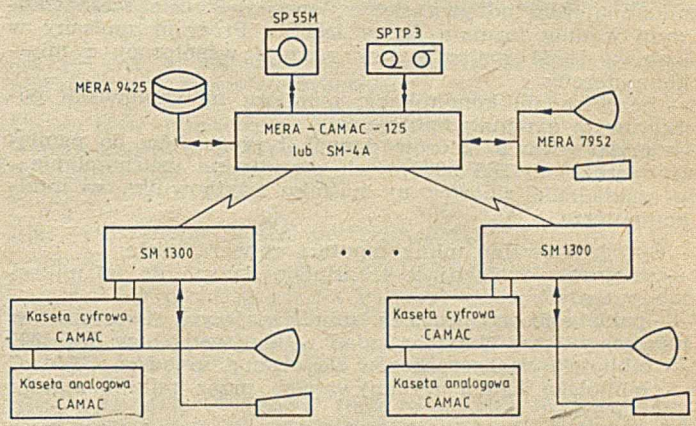
- uczy praktyki prowadzenia eksperymentów fizycznych i metod przetwarzania wyników
- wyrabia umiejętność posługiwania się środkami nowoczesnej aparatury pomiarowej i techniki obliczeniowej.

Organizacja laboratorium jest na tyle elastyczna, że umożliwia rozszerzenie zakresu wykonywanych pomiarów oraz ich samodzielne oprogramowywanie przez użytkownika.

KONFIGURACJA SPRZĘTOWA

Każde stanowisko pomiarowe jest wyposażone w aparaturę CAMAC umieszczoną w kasecie sterowanej przez mikrokomputer SM 1300 za pomocą sterownika typu 106A. Mikrokomputery są połączone z centralnym minikomputerem MERA-CAMAC-125 lub SM-4A, który pracuje pod kontrolą systemu operacyjnego DOS RW i zapewnia:
— przekazywanie programów i danych do mikrokomputerów SM 1300
— tworzenie zbiorów danych pomiarowych
— dostęp do oprogramowania użytkowego.

Blokowy schemat konfiguracji sprzętowej laboratorium przedstawiono na rysunku 1. W skład podstawowego wyposażenia minikomputera wchodzą:
— dwie stacje dysków sztywnych (MERA 9425)
— stacja dysków elastycznych (SP55M)
— stacja taśmy papierowej (SPTP3, SM 6204)
— konsola systemowa (DZM 180 KSR)
oraz urządzenia dodatkowe, jak np.:
— stacja pamięci taśmowej (IZOT 5003)
— drukarka wierszowa (DW-3M, DZM 180).



Rys. 1. Konfiguracja sprzętowa laboratorium

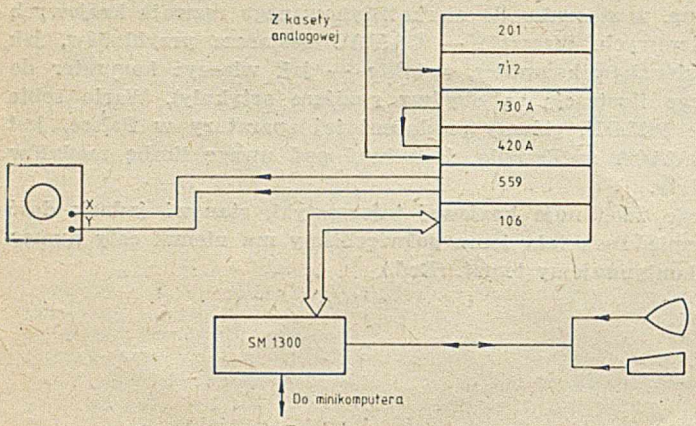
Mikrokomputery SM 1300 (do 15 szt.) są połączone z minikomputerem w sieć terminalową łączami transmisji szeregowej (V24), przy czym żaden z nich nie jest jednostką samodzielną, gdyż nie ma pamięci masowej.

Standardowa kaseta CAMAC jest wyposażona (rys. 2) w sterownik, stanowiący blok sprzężenia z mikrokomputerem oraz podstawowe moduły cyfrowe:
— impulsowy przetwornik analogowo-cyfrowy (712)
— blok sterowania obrazowaniem (559)
— pamięć o pojemności 1 K słów 24-bitowych (201)
— liczniki nastawne (420A).

Użycie wymienionych bloków pozwala na zrealizowanie podstawowych funkcji wielu typowych eksperymentów fizycznych, jak np. dokonanie pomiarów amplitudy impulsów w układzie analizatora, próbkowanie sygnałów i gromadzenie danych w układzie wieloprzelicznikowym [1].

[1]) Bliższe informacje na temat obsługi eksperymentów fizycznych można znaleźć w numerze monograficznym poświęconym systemowi CAMAC, INFORMATYKA, nr 4—5, 1982 — przyp. red.

Dzięki użyciu pamięci ferrytowej unika się straty danych w przypadku awarii zasilania mikrokomputera.



Rys. 2. Wyposażenie stanowiska pomiarowego

Dodatkowe wyposażenie kasety w moduły cyfrowe, takie jak bramki oraz rejestry wejściowo-wyjściowe, umożliwia wymianę praktycznie dowolnych danych doświadczalnych i współpracę kasety CAMAC z większą aparaturą pomiarową. Istotną częścią każdego zestawu są bloki analogowe, nie mające połączenia z magistralą kasety — umożliwiają one budowę wielu specjalizowanych torów pomiarowych.

OPROGRAMOWANIE LABORATORIUM

Minikomputer jako jednostka centralna nadzoruje proces tworzenia programu do obsługi eksperymentu oraz wspomaga opracowanie wyników dla jednostek podrzędnych (SM 1300). Jest to możliwe dzięki zapewnieniu bezpośredniej komunikacji operatora stanowiska pomiarowego z minikomputerem przez program zapisany w pamięci ROM mikrokomputera. Komunikacja pomiędzy SM 1300 a minikomputerem odbywa się znakowo w kodzie szesnastkowym według formatu firmy INTEL. Program zapisany w pamięci ROM realizuje trzy rodzaje współpracy z minikomputerem:
— bezpośrednią komunikację monitora na stanowisku pomiarowym z minikomputerem
— przepisywanie zbiorów danych i programów do pamięci operacyjnej SM 1300 i ich ewentualne uruchamianie
— zakładanie zbiorów na nośniku użytkownika w minikomputerze.

Oprogramowanie minikomputera zawiera także:
— programy obsługujące transmisję danych do (z) mikrokomputerów
— pakiet obsługi poleceń umożliwiających tworzenie w minikomputerze obrazów zadań wykonywanych w SM 1300
— bibliotekę makrorozkazów do obsługi urządzeń CAMAC
— bibliotekę zadań wykonywanych przez mikrokomputer SM 1300
— pakiet programów umożliwiających prowadzenie matematycznej analizy danych (operacje macierzowe, całkowanie funkcji przestępnych, analiza Fouriera, obliczenia statystyczne, generatory rozkładów losowych).

Przedstawione środki programowe oraz sprzętowe zapewniają realizację szerokiej gamy ćwiczeń, stwarzając jednocześnie możliwość rozbudowy laboratorium o programy obsługi eksperymentu i procedury przetwarzania danych.

PRZYKŁADOWA REALIZACJA STANOWISK LABORATORIUM STUDENCKIEGO

Laboratorium opracowano i wprowadzono do procesu dydaktycznego w Zakładzie Elektroniki Jądrowej i Medycznej Politechniki Warszawskiej, dlatego technika i fizyka jądrowa dominuje w dotychczas zrealizowanych eksperymentach. W zależności od wyposażenia w detektory promieniowania, odpowiednią aparaturę pomiarową i oprogramowanie, możliwe jest wykonanie wielu różnorodnych ćwiczeń, jak np.:
— organizacja i zasady pracy aparatury kontrolno-pomiarowej CAMAC
— optymalizacja pasma przenoszenia toru spektrometrycznego
— cyfrowe metody filtracji sygnałów analogowych
— detekcja słabych sygnałów świetlnych
— statystyka pomiarów przy rejestracji promieniowania jądrowego
— spektrometr Mössbauera, zasada i możliwości pomiarowe
— automatyczna analiza widm promieniowania jonizującego
— badanie liczników proporcjonalnych
— pomiar radioaktywnego zanieczyszczenia materiałów budowlanych
— analiza fluorescencyjna lub aktywacyjna
— pomiar okresu połowicznego rozpadu
— analiza sygnałów EKG i innych
— detekcja sygnałów z układu bodźcowo-przewodzącego serca.

Przykładowo — celem ćwiczenia pn. „Statystyka pomiarów przy rejestracji promieniowania jądrowego" jest zapoznanie się z kilkoma ważnymi w technice jądrowej rozkładami zmiennych losowych. Wykonanie ćwiczenia polega na zarejestrowaniu empirycznych rozkładów tych zmiennych oraz weryfikacji (przy użyciu testu χ^2) hipotez dotyczących rodzaju rozkładów.

Każdorazowo po przeprowadzonym pomiarze student ma możliwość dokonania normalizacji zmierzonego rozkładu, jego zobrazowania, obliczenia momentów i dokonania weryfikacji rozkładu przy użyciu wybranego testu zgodności. Wyniki pośrednie są prezentowane na terminalu użytkownika, a dokumentacja wyników ćwiczenia jest prowadzona na wybranym nośniku minikomputera.

* * *

Ćwiczenia są realizowane na tym samym sprzęcie, a różnice sprowadzają się do oprogramowania i doboru urządzeń wejściowych sygnału. Ponieważ w składzie typowego wyposażenia kasety cyfrowej znajdują się wszystkie użyte w ćwiczeniach bloki, ćwiczenia mogą być prowadzone równolegle (rozłącznie czasowo). Przez zmianę oprogramowania tor pomiarowy zbudowany do jednego ćwiczenia może służyć jeszcze do przeprowadzania wielu innych.

Przedstawiona konfiguracja laboratorium umożliwia:

- zmniejszenie kosztu aparatury użytej do realizacji ćwiczeń, co jest skutkiem elastyczności oprogramowania, ale także skutkiem skoncentrowania urządzeń peryferyjnych przy minikomputerze, dzięki czemu zaoszczędza się ok. miliona zł na każdym mikrokomputerze, wskutek wyeliminowania pamięci dyskietkowej
- zmniejszenie liczby studentów pracujących na danym stanowisku laboratoryjnym, gdyż dane ćwiczenie może być prowadzone równolegle na wielu stanowiskach, oraz zwiększenie aktywności studenta w czasie przebiegu ćwiczenia
- koncentrację studentów na merytorycznej treści ćwiczenia i odciążenie ich od tworzenia dokumentacji
- skuteczne wprowadzenie nowoczesnej techniki pomiarowej i obliczeniowej do dydaktyki na uczelniach.

Laboratorium może być wykorzystane na wydziałach elektrycznych i elektronicznych w pracowniach teorii obwodów i sygnałów, automatyki, maszyn elektrycznych, robotroniki, techniki biomedycznej, a także na innych wydziałach technicznych, jak również fizycznych, biologicznych itp.

Mikroprocesor 8085 jest udoskonaloną wersją 8080. Architekturą i możliwościami nie dorównuje jednak Z80 i z tego względu jest — na świecie — znacznie rzadziej stosowany. Rozszyfrowanie dodatkowych, nie publikowanych przez producenta rozkazów zwiększa nieco efektywność programowania, ale nowej gwiazdy niestety nie kreuje. W Polsce „specjalizujemy się" w układach firmy INTEL i choćby były gorsze i droższe — jakaś magiczna siła zamienia skąpe ilości dolarów na produkty tej firmy lub ich odpowiedniki. Znając zaś prawidłowości naszej gospodarki i chcąc uniknąć iluzji na łamach mikroKLANU, nie zamierzamy walczyć z tą tendencją. Stąd też wynika przeświadczenie, że informacja o odkrytych (bynajmniej nie w Polsce) nowych rozkazach dla mikroprocesora 8085 będzie przydatna dla wielu mikroKLANOWCÓW.

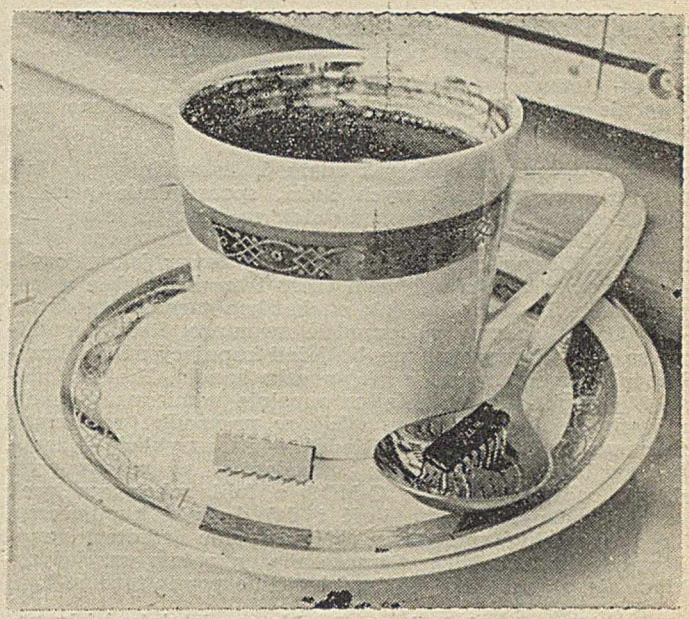
Nieznane rozkazy mikroprocesora 8085

Wśród 256 możliwych kodów rozkazów mikroprocesora 8080, dwanaście to kody niewykorzystane nie występujące w oficjalnych materiałach firmowych. Mikroprocesor 8085 wykorzystuje dodatkowo dwa z tych kodów dla rozkazów RIM i SIM (opisane w materiałach firmowych i podręcznikach firmy INTEL).

Z niewyjaśnionych przyczyn znaczenie pozostałych dziesięciu kodów nie zostało dotąd opublikowane przez producenta. Na podstawie opracowanych testów dla mikroprocesorów w wersji 8085 i 8085A odkryto znaczenie tych dziesięciu kodów rozkazów, a także znaczenie dwóch bitów w rejestrze stanu. Siedem spośród tych dziesięciu rozkazów dotyczy operacji na danych 16-bitowych, a trzy pozostałe wykorzystują „nowe" bity stanu. Proponowane mnemoniki dla nowych rozkazów są zgodne z regułami stosowanymi przez firmę INTEL. W tabeli zestawiono kody nowych rozkazów, ich mnemoniki oraz skrócony opis działania.

Nieznane operacje mikroprocesora Intel 8085

Kod operacji (hex.)	Mnemonik	Zmienia bity stanu	Liczba cykli		Skrócony opis działania
			maszynowych	zegarowych	
08	DSUB	Z, S, P, CY, AC, X5, V	3	10	(H)(L) = (H)(L) − (B)(C)
10	ARHL	CY	2	7	(H7) = (H7), (Hn−1) = = (Hn), (L7) = (H0), (Ln−1) = (Ln), (CY) = = (L0)
18	RDEL	CY, V	3	10	(Dn+1) = (Dn), (D0) = = (E7), (CY) = (D7), (En+1) = (En), (E0) = (CY)
28	LDHI	–	3	10	(D)(E) = (H)(L) + (bajt 2)
38	LDSI	–	3	10	(D)(E) = (SPH)(SPL) + + (bajt 2)
CB	RSTV	–	1/3	6/12	(V = 1) → (SP) −1 = (PCH), ((SP)−2) = (PCL), (SP) = (SP)−2, (PC) = = 40H
D9	SHLX	–	3	10	((D)(E)) = (L), ((D)(E)+1) = (H)
DD	JNX5	–	2/3	7/10	(X5 = 0) → (PC) = = (bajt 3)(bajt 2)
ED	LHLX	–	3	10	(L) = ((D)(E)) (H) = ((D)(E)+1))
FD	JX5	–	2/3	7/10	(X5 = 1) → (PC) = = (bajt 3)(bajt 2)



Człowiek z Perskiego

µK: — Jak to się stało, że sprzedajesz na Perskim Jarmarku akurat części elektroniczne?

C: — Dość prozaicznie. Niektórych elementów miałem więcej, innych potrzebowałem... I tak zacząłem przychodzić na Perskł. Ku mojemu zdziwieniu spotkałem tam wielu znajomych z Uczelni.

µK: — Z Uczelni? Jesteś profesjonalistą?

C: — Tak, skończyłem Wydział Elektroniki. Ale studia nie dają wiedzy praktycznej. Na Perskim przez pół roku zdobyłem chyba więcej wiadomości niż przez pięć lat Politechniki. Głównie dlatego, że zacząłem coś robić samemu i mogłem to skonfrontować z innymi praktykami.

µK: — Co daje Ci Perski? Dużo zarabiasz? Jeździsz własnym samochodem w odróżnieniu od innych absolwentów?
C: Nie, takich pieniędzy na tym nie robię, samochodu nie mam. No, stać mnie czasem na taksówkę czy filiżankę kawy, ale chyba nie na tym polega luksusowe życie? Handlu na dużą skalę nie prowadzę, bo mnie to nie bawi; zresztą chyba nie potrafię... W Perski po prostu wsiąkłem.

µK: — Zaobserwowałem tam niecodzienną atmosferę. Nieznajomi wymieniają doświadczenia jak bliscy koledzy. Sprzedający zupełnie nie przypominają sklepowej ekspedientki. Rozmawiają, radzą... Widziałem — na przykład — jak doradzano kupującemu, oferując mu tańsze elementy niż początkowo chciał zakupić.

C: — To jest logiczne! Człowiek zajmuje się tym, na czym się zna. Praktycznie wszyscy handlujący na Perskim znają się dokładnie na tym, co sprzedają, bo prawie każdy z nas jest hobbystą. Oczywiście, jak w każdym środowisku, tak i na Perskim jest pewna grupa ludzi, którzy zajmują się tym wyłącznie dla handlu — mają tzw. dojścia i mogą na tym zarobić. To jest 10, może 20% ogółu. Wyspecjalizowanych hobbystów jest jednak znacznie więcej, co widać zresztą po asortymencie: ktoś ma tylko „analogówkę", inny „cyfrówkę", a ostatnio pojawili się specjaliści od mikroprocesorów. Co więcej — oferowane są też półprodukty i gotowe urządzenia własnego wyrobu: wzma-

Człowiek z Perskiego

cniacze, zasilacze, elektroniczne pozytywki czy zegary. Niedawno ktoś oferował płytki prostego mikrokomputera.

µK: — Gotowe urządzenia... Jaka jest gwarancja, że kupujący nie zostanie oszukany? Przecież z reguły nie ma możliwości sprawdzenia towaru na miejscu.

C: — Zapewne może się tak zdarzyć. Szczególnie, gdy kupuje się od przypadkowych ludzi, sporadycznie pokazujących się na Perskim i na dodatek za okazyjną cenę. Bywalcy Perskiego na taką kompromitację nie mogą sobie pozwolić. Oczywiście, należy zakupiony towar jak najszybciej sprawdzić, bo z upływem czasu transakcja idzie w zapomnienie. Na Perskim, jak na każdym targu, niezadowolony klient ma p r a w o urządzić awanturę. Rzeczy wadliwe są z reguły wymieniane, bo jeżeli kupujący narobi krzyku, to sprzedający ma z głowy cały handel. Nikt od niego nie kupi... Tu ma miejsce zupełnie odmienna sytuacja niż w oficjalnym handlu, gdzie w przypadku reklamacji zawsze winny jest klient. Kiedyś — na przykład — zaobserwowałem taki kwiatek: sprzedawca, w państwowym sklepie, odwinął układy C-MOS z folii zabezpieczającej, odliczył palcami żądaną ilość i dał do ręki kupującemu. Trudno tu winić sprzedawcę, który pewnie przedtem sprzedawał ziemniaki. Niby dlaczego ma on potem przyjąć reklamację, skoro układy były fabrycznie przetestowane [1]).

µK: — Wracając jednak na Perski: czy istnieje ścisły podział na kupujących i sprzedających?

C: — Często zdarzają się wymiany między handlarzami, ten potrzebuje tego, tamten czegoś innego. To są przecież ludzie, którzy się elektroniką zajmują. Dlatego też nie warto oszukiwać. Grozi to utratą wiarygodności i kontaktów. Kiedyś zdarzyła mi sią taka sytuacja: na 80 wzmacniaczy operacyjnych — 20 to były tylko wyprowadzenia zalane żywicą. Oczywiście nabywca wrócił z reklamacją, lecz ja nie miałem już innych na wymianę. Pomógł mi sąsiad, który trzy stoły dalej sprzedawał takie same wzmacniacze, ale o 20 zł drożej. Specjalnie dla mnie obniżył cenę, tak aby mój klient odszedł usatysfakcjonowany. A wracając jeszcze do podziału na dwie strony lady... Był taki moment, że dwóch chłopaków z technikum regularnie przez kilka tygodni zamęczało mnie pytaniami o obudowę zasilaczy: który „scalak" lepszy, jakie stosować oporniki, itd. Teraz sprzedają na Perskim zasilacze.

µK: — Czy nie wylania nam się tutaj zbyt sielankowy obraz Perskiego?

C: — Należy chyba zacząć od stwierdzenia, że oficjalne ceny półprzewodników są absurdalne w porównaniu z płacami. Jeśli liczyć nawet po 700 zł za dolar, a to i tak układy zachodnie są często tańsze (nie mówię tu oczywiście o cenach ze sklepu w „Intraco" [2])! Na Perskim zdarzają się niestety towary kupione w sklepach czy w BOMIS-ie — ja to osobiście potępiam jako zwyczajną spekulację. Są też rzemieślnicy montujący telewizory z niepełnowartościowych podzespołów (czasem o lepszej jakości niż fabryczne). Tak czy inaczej — Perski to bazar, po którym trzeba nauczyć się poruszać.

µK: — Jakie masz rady dla początkujących?

C: — Trzeba przyjść, obejrzeć. Potem drugi raz, trzeci. Porozmawiać kto co ma. Unikać okazji i ludzi sprzedających raptem trzy elementy. Trzeba po prostu poznać dziedzinę, w której chce się coś robić...

µK: — Przewija się ciągle w tej rozmowie edukacyjna rola Perskiego. Może widzisz w tym jakąś misję?

---

[1]) Dotknięcie palcem końcówki układu C-MOS z reguły powoduje jego zniszczenie przez ładunek elektrostatyczny — przyp. AJP

[2]) W budynku „Intraco" w Warszawie ma swój sklep firma UNITEX, sprzedająca za waluty wymienialne podzespoły i urządzenia produkowane na Zachodzie — przyp. AJP

DSUB — odejmowanie dwóch liczb 16-bitowych zawartych w rejestrach HL i BC; komplementarne z instrukcją DAD B

ARHL — przesunięcie arytmetyczne w prawo o jeden bit zawartości pary rejestrów HL poprzez bit przeniesienia, z powieleniem najstarszego bitu w rejestrze H

RDEL — rozszerzona na liczbę 16-bitową operacja przesunięcia cyklicznego w lewo o 1 bit (poprzez bit przeniesienia) zawartości pary rejestrów DE; analogiczna do operującej na liczbach 8-bitowych instrukcji RAL

LDHI — umożliwia realizację adresowania indeksowego poza 256-bajtową granicę strony; ładuje do DE zawartość HL powiększoną o podany bajt

LDSI — nadaje się w szczególności do operacji na stosie w różnych podprogramach; ładuje do DE wskaźnik stosu powiększony o podany bajt

RSTV — działa jako warunkowy rozkaz RESTART od adresu 40H w przypadku, gdy bit V w rejestrze stanu jest ustawiony na 1; w przeciwnym razie działa jak instrukcja NOP; bit V wskazuje na przepełnienie w trakcie wykonywania operacji na liczbach przedstawionych w uzupełnieniu dwójkowym

SHLX — zapamiętanie zawartości pary rejestrów HL w komórce o adresie wskazanym przez parę rejestrów DE

JX5, JNX5 — reagują na bit X5 analogicznie jak JC i JNC na bit przeniesienia CY i mogą być uważane jako warunkowe rozkazy skoku; bit X5 nie odpowiada żadnemu ze znanych standardowych bitów stanu, lecz określony jest przez swoją pozycję w rejestrze stanu — znajduje on zastosowanie jako wskaźnik przepełnienia dla liczb bez znaku w operacjach typu INX i DCX, w szczególności przy przejściu z FFFFH na 0000 (INX) lub z 0000 na FFFFH (DCX)

LHLX — ładowanie pary rejestrów HL zawartości komórki o adresie wskazanym przez parę rejestrów DE.

Wykorzystanie „nowych" rozkazów w programach napisanych w języku maszynowym następuje analogicznie do „starych" — w kolejnych komórkach pamięci umieszczony jest kod operacji i wymagana liczba bajtów z danymi. W przypadku współpracy z programem ASSEMBLER, który nie rozpoznaje użytych kodów mnemonicznych, najlepiej jest zdefiniować odpowiednie makroinstrukcje, które ustawiałyby kod nowego rozkazu i ewentualne argumenty jako bajty danych.

Nowe rozkazy nadają się w szczególności do wykorzystania w podprogramach arytmetycznych. Przykładowo — dzielenie liczby 16-bitowej przez liczbe 16-bitową z użyciem rozkazów RDEL i DSUB trwa średnio ok. 368 µs a mnożenie 278 µs. Programy takie są około dwukrotnie szybsze niż w przypadku nie stosowania tych nowych rozkazów.

Przykładowe programy pokazują wykorzystanie nowych rozkazów.

Przykład 1

```
; MUL 16- MNOŻENIE DWÓCH DODATNICH LICZB 16-BITOWYCH
; WYWOŁANIE : CALL MUL16
; PARAMETRY : BC - MNOŻNA (MSB, LSB)
;             DE - MNOŻNIK (MSB, LSB)
; WYNIK : 32 BITY W REJESTRACH DE (MSB), HL (LSB)
MUL16: LXI   H,0      ; WYZEROWANIE PARY HL
       MVI   A,17     ; INICJOWANIE LICZNIKA BITÓW
MULA:  DCR   A        ; MNOŻENIE SKONCZONE?
       RZ             ; TAK, POWROT Z PODPROGRAMU
       DAD   H        ; WYNIK JEDEN BIT W LEWO
       RDEL           ; NOWY ROZKAZ - DE 1 BIT W LEWO
       JNC   MULA     ; BIT MNOŻNIKA = 1?
       DAD   B        ; TAK, DODAJ MNOZNA
       JNC   MULA     ; NIE MA PRZEPELNIENIA,
                      ; NASTĘPNY BIT
       INX   D        ; PRZEPELNIENIE Z HL DO DE
       JMP   MULA     ; NASTEPNY BIT
```

Przykład 2

```
; DIV32 – DZIELENIE LICZB ZNAKOWANYCH DODATNICH
; WYWOLANIE : CALL DIV32
; PARAMETRY : DE (MSB), HL (LSB) – DZIELNA 32-BITY (MSBit = 0)
;             BC (MSB) – DZIELNIK 16-BITOW (MSBit = 0)
; WYNIKI    : DE (MSB, LSB) – ILORAZ – 16-BITOW
;             : HL (MSB, LSB) – RESZTA
DIV32: XCHG                 ; WYMIANA ZAWARTOSCI
                            ; REJESTRÓW DE I HL
; BADANIE WYSTAPIENIA NADMIARU PRZY DZIELENIU
       DSUB                 ; CZY WYSTAPI NADMIAR?
       JNC   DIVOV          ; TAK, SKOCZ DO PROGRAMU
                            ; OBSŁUGI
; NIE WYSTAPI NADMIAR

       DAD   B      ; ODTWORZENIE MSB DZIELNEJ
       MVI   A,17   ; INICJOWANIE LICZNIKA BITÓW
DIVA:  DCR   A      ; DZIELENIE SKONCZONE?
       RZ           ; TAK, POWROT Z PODPROGRAMU
       DAD   H      ; PRZESUN MSB DZIELNEJ O 1 BIT W LEWO
       ANI   0FFH   ; USTAW CY = 0
       RDEL         ; PRZESUN LSB DZIELNEJ O 1 BIT W LEWO
                    ; (E0 = 0)
       JNC   DIVE   ; D7 = 1?
       INR   L      ; TAK, POWIEKSZ MSB DZIELNEJ
DIVE:  DSUB         ; UTWORZ RESZTE CZESCIOWA
       JC    DIVD   ; WYNIK ODEJMOWANIA UJEMNY?
       INR   E      ; NIE, TWORZ ILORAZ (E0 = 1)
       JMP   DIVA   ; NASTEPNY BIT
DIVD:  DAD   B      ; TAK, ODTWORZ RESZTE CZESCIOWA
       JMP   DIVA   ; NASTEPNY BIT
; PROGRAM OBSŁUGI NADMIARU PRZY DZIELENIU
DIVOV: .....
```

LITERATURA

[1] Dehnhardt W., Sorensen V. M.: Unbekannte 8085 Instruktionen. Elektronik, nr 15/1978, s. 66
[2] Dehnhardt W., Sorensen V. M.: Unspecified 8085 op codes enhance programming. Electronics, January 18, 1979, s. 144–145
[3] MCS-80 User's Manual, Intel, October 1977.

ZBIGNIEW BARBAŚ
Ośrodek Informatyki
Politechnika Poznańska

CENY ● CENY ● CENY ● CENY ● CENY

grudzień 1983

Podzespoły

USA ($):

Z80 — 4, Z80A — 5, Z80B — 10; 8080 — 3, 8085 — 5, 8085A-2 — 7, 8086 — 25; 6800 — 3, 6809E — 15; 68000 — 50; 6502 — 5, 6502A — 7, 6502B — 10; Z8001 — 45, Z8002 — 35; 8 × 4116 (120 ns) — 30, 8 × 4116 (200 ns) — 13; 4164 (150 ns) — 7, 4164 (200 ns) — 6;2716 (350 ns) — 6, 2716 (450 ns) — 4; 2732 (200 ns) — 12, 2732 (250 ns) — 9, 2732 (450 ns) — 5, 2764 (200 ns) — 25, 2764 (450 ns) — 10

Polska — UNITEX ($):

Z80A — 10,2, Z80A PIO — 11,4, Z80 PIO — 9,8, Z80A CTC — 11,4, Z80 CTC — 9,8, Z80A DMA — 33,6, Z80 DMA — 27,2, Z80A SIO — 37, Z80 SIO/2 — 28,5, Z80A DART — 27,2, Z80 DART — 23,3; 2716/4 — 10,2; 2114 — 5,3; 4116 — 4,3

RFN (DM):

Z80A — 8,5; 8080 — 9,5, 8085 — 10,3, 8086 — 65; 68000 — 220; 2716 — 9,8, 2732 — 16, 2764 — 20; 4116 (200 ns) — 2,9, 4164 (200 ns) — 14,8

Wielka Brytania (£)

Z80A — 2,43; 8080 — 2,5, 8085 — 3,5, 8086 — 18; 68000 — 78; 2716 — 2,25, 2732 — 3,5, 2764 — 5,9; 4116 — 0,75, 4164 — 3,5.

Człowiek z Perskiego

C: — Ludzie powinni uczyć się w szkole, na uczelni, z książki i czasopism fachowych. Perski to jedynie korepetycje. W niektórych dziedzinach znacznie poszerzone, wobec braku odpowiedniej literatury. Tu rządzą prawa rynku. Na przykład ci, którzy zajmują się mikroprocesorami, oferują zazwyczaj odbitki z katalogów, noty aplikacyjne, schematy... Rzeczy nieosiągalne w księgarniach. To może dowód absurdu, że handlarz z Perskiego lepiej wie czego ludzie potrzebują, niż wydawnictwa.

µK: — Czy nie mając — cioci w Ameryce, a mając jako zaplecze jedynie Perski, można zabrać się za budowę własnego mikrokomputera?

C: — Trzeba najpierw dysponować pewną wiedzą i doświadczeniem. Nie radzę nikomu zaczynać od dużych inwestycji. Ale prosty systemik — CPU, trochę RAM, EPROM, wyświetlacze na LED-ach — można skompletować za mniej niż 20 tys. Potem można go rozbudowywać. I na pewno warto — sądzę, że mikrokomputer o walorach ZX SPECTRUM można zbudować samemu za ok. 100 tys. zł.

µK: — A czy istnieje u nas rynek dla oprogramowania?

C: — Ciągle jeszcze w powijakach. Zbyt mało mikrokomputerów znajduje się w prywatnych rękach. W Polsce jest odwrotna sytuacja niż na Zachodzie — najdroższy jest sprzęt, a oprogramowanie można zawsze jakoś wykombinować. Ale zauważyłem, że kilku ludzi oferujących programy na ZX81 i SPECTRUM zaczęło już w miarę regularnie pojawiać się na Perskim. A zatem rynek taki zapewne niebawem powstanie, tym bardziej, że same układy można połączyć raptem na kilka sposobów — i na tym koniec. Dopiero oprogramowanie ożywia komputer.

µK: — Czy zdarza się, że przychodzi na Perski prywaciarz, który chce wykorzystać komputer „w interesie"?

C: — Mam właśnie przykład z ostatniej soboty. Do kolegi zwrócił się klient: „Chciałbym zautomatyzować pewien proces technologiczny, czy mógłby tym sterować mikroprocesor i jak by wyglądała cena takiego sterownika?

µK: — I dostał ofertę?

C: — Trudno było strzelać w ciemno, ale cena nie powinna przekroczyć 150—200 tys. zł. Klient stwierdził, że taka propozycja go interesuje. Wcześniej próbował w przedsiębiorstwach państwowych i firmach polonijnych. Ci pierwsi nie byli zainteresowani, ci drudzy ocenili inwestycję na co najmniej pół miliona.

µK: — Cyli na Perskim powstają zalążki nowych, bardziej efektywnych firm?

C: — Niezupełnie. W firmach polonijnych niekoniecznie siedzą tępaki. Gdyby kolega miał na utrzymaniu Urząd Podatkowy, księgowego i kilku kontrolerów, to chyba taniej by mu to nie wypadło.

µK: — A więc jest to w gruncie rzeczy działalność bez perspektyw!

C: — A czy Ty, jako redaktor mikroKLANU, masz jakieś wielkie perspektywy? Wszyscy jedziemy na tym samym wózku. Takie czasy.

µK: — Jeszcze jedno pytanie: czy masz do kogoś, o coś żal? Może o te długie godziny spędzone w upale lub na mrozie?

C: — Nie! Zrobiono bardzo dużo dając nam miejsce do spotkań. Nawet inwigilacja (wiadomo, że musi być) jest robiona raczej dyskretnie.

Tylko szkoda, że nowoczesne podzespoły to z importu, że informacja to kserokopie z zachodnich wydawnictw, a mikrokomputer kosztuje ponad dziesięć pensji, a nie jedną dziesiątą jak w krajach, gdzie na „pchlim targu" nie handluje się elektroniką.

Rozmawiał
ANDRZEJ J. PIOTROWSKI
Fot. A. Piastka

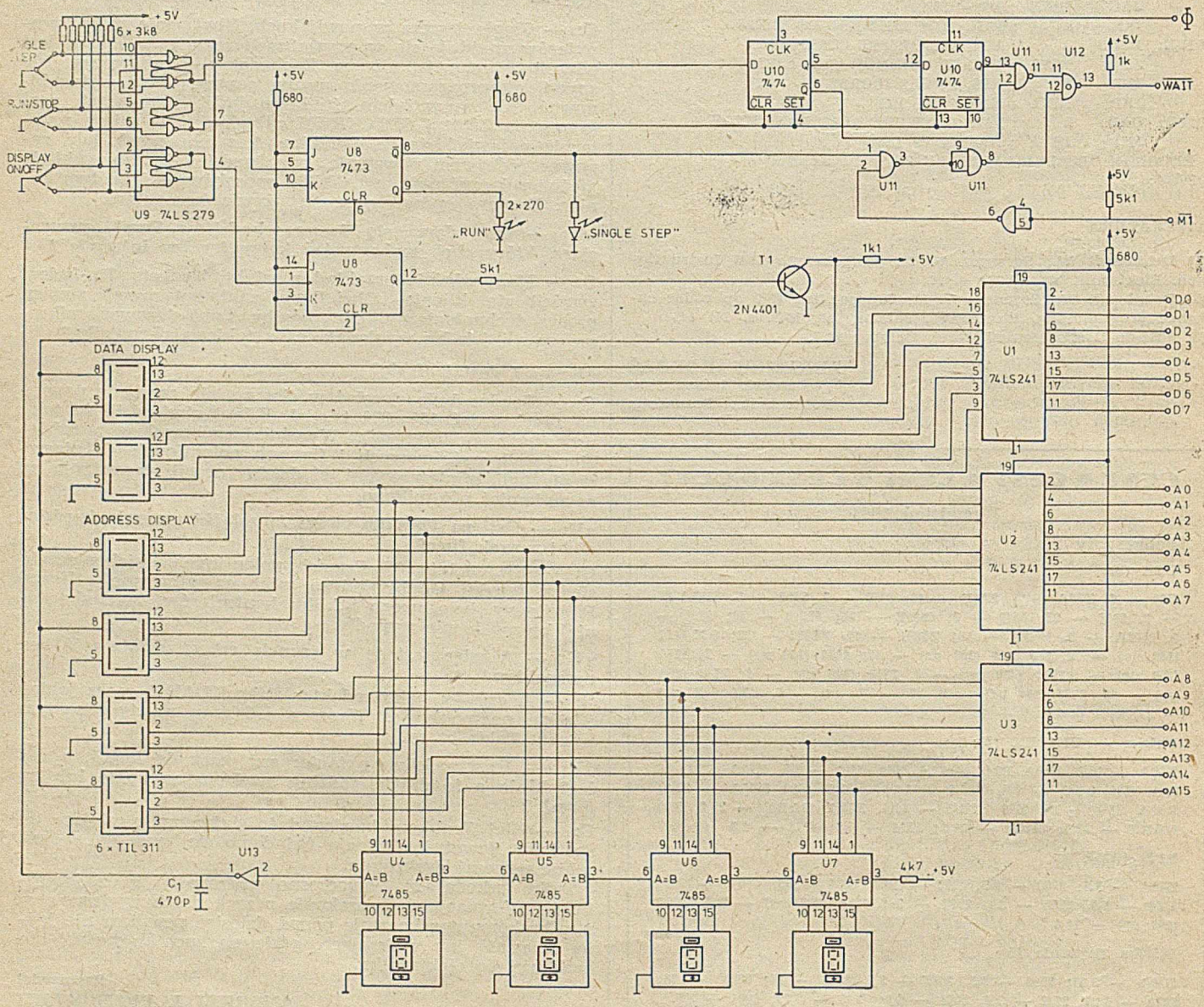
W pierwszym wydaniu MikroKLANU opisana została programowa realizacja pracy krokowej. Dla tych co wolą „plątać się w drutach" zamieszczamy rozwiązanie sprzętowe. Do jego niewątpliwych zalet należy możliwość „kroczenia przez program" zawarty w pamięci ROM. Dzięki temu można podpatrzyć niektóre tajemnice producentów mikrokomputerów lub zlokalizować błąd w tej pamięci. Układ można również wykorzystać przy uruchamianiu systemu nie wyposażonego w monitor ekranowy.

Koncepcja rozwiązania została zaczerpnięta z czasopisma „Elektronik" (kwiecień 1983), opisaliśmy jednak też sposób zaadaptowania jej do polskich warunków.

Układ pracy krokowej dla Z80

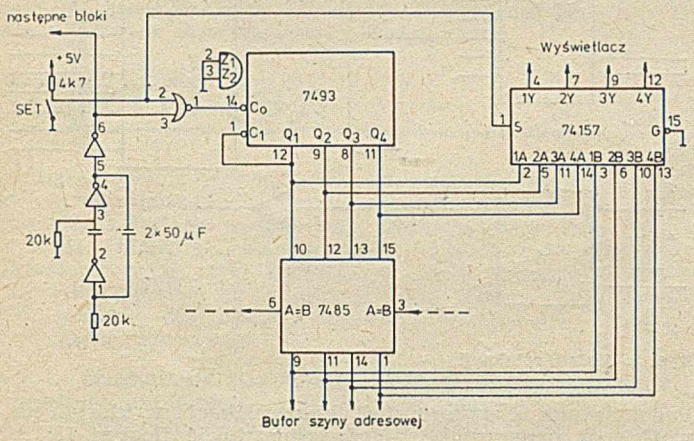
Prezentowany układ wstrzymuje pracę mikroprocesora po wykryciu zadanego kodu na szynie adresowej. Możliwe jest również ręczne wstrzymanie przez przyciśnięcie klawisza RUN/STOP. Po zatrzymaniu procesora można kontynuować program realizując krokowo kolejne rozkazy.

Adres, pod którym ma być wstrzymana realizacja programu, ustawiany jest za pomocą specjalnych przełączników rotacyjnych, wymuszających na czterech liniach wyjściowych kod dwójkowy ustawionej cyfry. Linie wyjściowe doprowadzone są do komparatorów (U4, U6 i U7), które dokonują porównania zadanych kodów z informacją znajdującą się na szynie adresowej. W przypadku stwierdzenia zgodności przez wszystkie cztery komparatory, zerowany jest przerzutnik U8. Kondensator C1 (rys. 1) służy do odfiltrowania krótkich impulsów (ang. glitch), mogących powstać przy zmianach wysterowania szyny adresowej. Sygnał z przerzutnika U8 poprzez bramki U11 i U12 podawany jest na linię WAIT i wymusza przejście procesora w stan oczekiwania. Przyciśnięcie klawisza „SINGLE STEP" zwalnia okresowo blokadę procesora, co umożliwia wykonanie jednego rozkazu. Przyciśnięcie klawisza RUN/STOP powoduje odblokowanie linii WAIT. Powrót do pracy krokowej możliwy jest przez ponowne przyciśnięcie klawisza RUN/STOP.



Rys. 1. Schemat układu pracy krokowej dla Z80

Sygnały szyny danych i szyny adresowej są buforowane na wejściu układu (U1, U2 i U3), a następnie podawane na wyświetlacze 7-segmentowe o kodowaniu szesnastkowym (TIL-311).



Rys. 2. Układ elektronicznego przełącznika rotacyjnego

Niektóre układy zastosowane w prezentowanym rozwiązaniu są stosunkowo trudno osiągalne w kraju. Kosztem „elegancji" rozwiązania, a także zwiększenia poboru prądu ze źródła zasilania — można jednak dokonać adaptacji układu do polskich warunków. Bufory 74LS241 można z powodzeniem zastąpić polskimi odpowiednikami 8216, a nawet zwykłymi bramkami. Brak wyświetlaczy kodowanych szesnastkowo można ominąć, sterując zwyczajne wyświetlacze 7-segmentowe za pośrednictwem odpowiednio zaprogramowanych pamięci PROM o organizacji 32 × × 8 bitów. Tranzystor T1 należy dobrać w zależności od typu zastosowanych wyświetlaczy. Nietrudno też zastąpić układ 74LS279 stosując układy 74LS00 lub nawet 7400.

Nieco trudniej znaleźć substytut dla przełączników rotacyjnych. Można by zastosować niezależny przełącznik dla każdej linii (sterowanie binarne), ale jest to rozwiązanie wyjątkowo niedogodne dla użytkownika. Bardziej eleganckim, choć droższym, rozwiązaniem jest zastosowanie elektronicznych przełączników rotacyjnych. Na rysunku 2 przedstawiono schemat takiego rozwiązania (fragment dla jednej cyfry).

Zasadniczym elementem jest licznik binarny 7493 o równoległych wyjściach, które podawane są do komparatora oraz multipleksera szyny adresowej. Przyciśnięcie klawisza SET otwiera przejście dla impulsów zegara sterujących wejście licznika i równocześnie przełącza multiplekser adresów, tak aby na wyświetlacz podawane były sygnały z wejścia licznika. Dla podanych wartości elementów częstotliwość generatora wynosi ok. 2 Hz, co pozwala na szybkie ustawienie żądanej cyfry.

Opracował
A.J.P.

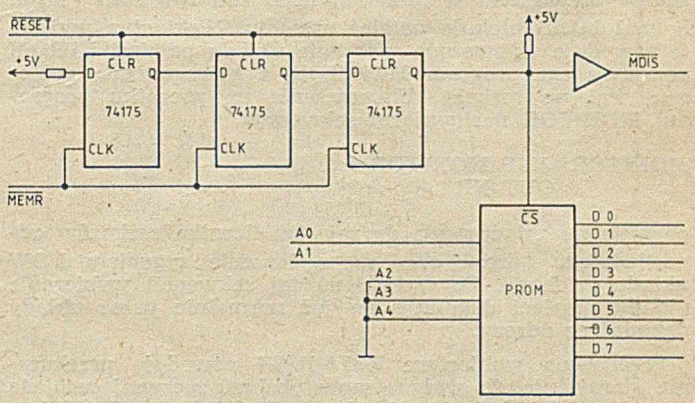
Oto kolejny przykład pokazujący, że stosowanie przestarzałej techniki zmusza do myślenia. Projektanci 16-bitowych mikroprocesorów wyeliminowali opisywany niżej problem, tak więc do dalszej lektury zapraszamy tych uparciuchów, którzy w mikrokomputerach stosują 8080.

W większości mikroprocesorów 8-bitowych podanie sygnału RESET powoduje wyzerowanie licznika rozkazów. Tym samym program powinien zaczynać się od adresu 0000H. Niby wszystko jest logiczne, jeśli przyjmiemy, że procesor wykonuje zawsze ten sam program zawarty w ROM-ie. Jeżeli jednak mamy do czynienia nie tyle z mikroprocesorowym sterownikiem, co z komputerem realizującym rozmaite programy wpisywane do RAM-u, to rozwiązanie logiczne okazuje się szalenie niewygodne (przerwania, umieszczanie tabel skoków, kompatybilność programów realizowanych na różnych systemach, itp). Stąd też powstał pomysł „oszukania" mikroprocesora, tak aby po sygnale RESET automatycznie przechodził do realizacji programu zarządzającego (zawartego w ROM-ie), umieszczonego pod koniec obszaru adresowego. Sposobów „oszukiwania" jest niemal tyle, ile systemów mikrokomputerowych. Są to jednak z reguły modyfikacje kilku podstawowych pomysłów. Dwa z nich przedstawiamy poniżej. Dla tych, którzy lubią fikuśne rozwiązania, warto dodać, że przedstawione pomysły można również wykorzystać dla obsługi przerwań wektorowych — generujących kody RST0...RST7.

Przekazywanie sterowania do programu MONITOR

Automatyczne przekazanie sterowania do programu zarządzającego po wystąpieniu sygnału RESET można zrealizować stosując proste rozwiązania sprzętowe.

Jako pierwsze przedstawiono rozwiązanie wykorzystujące dodatkową pamięć PROM (32 × 8 bajtów). W trzech początkowych komórkach pamięci zapisany jest rozkaz skoku do MONITORA (rys. 1). Sygnał RESET zeruje wyjścia przerzutników, co blokuje pamięć systemu (sygnałem $\overline{MDIS}$) oraz uaktywnia układ pamięci PROM. Trzy kolejne

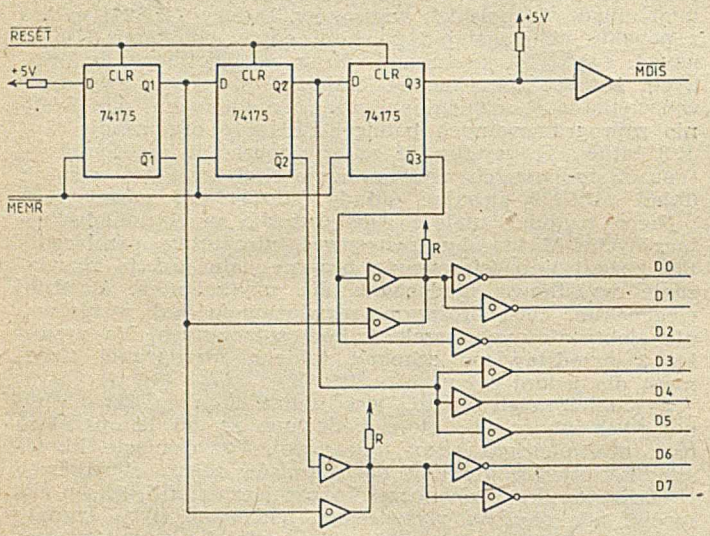


Rys. 1. Układ dostępu do programu zarządzającego po sygnale RESET z wykorzystaniem pamięci PROM

sygnały odczytu ($\overline{DBIN}$ lub $\overline{MEMR}$) powodują wprowadzenie i wykonanie przez procesor rozkazu skoku na początek programu MONITOR. Narastające zbocze trzeciego sygnału odczytu odblokowuje pamięć systemową. Pamięć PROM staje się nieaktywna do momentu podania kolejnego sygnału RESET. Wadą tego rozwiązania jest konieczność stosowania dodatkowej pamięci PROM. Układ można zmodyfikować wykorzystując bramki typu „otwarty kolektor" zamiast pamięci PROM.

Na rysunku 2 przedstawiono zmodyfikowaną wersję rozwiązania. W przykładzie założono, że program MONITOR rozpoczyna się od adresu F800H. W tabeli zilustrowano sposób generowania odpowiednich kodów na szynie danych. Wadą tego rozwiązania jest konieczność zmiany konfiguracji bramek dla różnych adresów początkowych programu MONITOR.

Całkowicie odmienne rozwiązanie przedstawione zostało na rysunku 3. Zastosowano tutaj „układ relokacji". Cechą charakterystyczną tego rozwiązania jest dekodowanie adresów programu zarządzającego w dwóch obszarach. Po



Rys. 2. Układ dostępu do programu zarządzającego po sygnale RESET z wykorzystaniem bramek otwarty kolektor

sygnale RESET jest to obszar od adresu 0000H, a po dokonaniu relokacji — obszar, w którym umieszczony jest program MONITOR. W odróżnieniu od poprzednich rozwiązań na początku programu MONITOR umieszczony jest skok do następnego rozkazu programu. Sygnał RESET ustawia na wyjściu Q (przerzutnik D lub RS) stan niski. Sygnał MDIS blokuje pamięć systemu. Niski stan wyjścia Q powoduje wymuszone wybranie układu pamięci EPROM zawierającego program MONITOR, z którego będą pobierane kolejne rozkazy. Umieszczenie na początku programu MONITOR następującej sekwencji:

```
MONITOR: JMP MONITOR + 3
         OUT NR
```

pozwala na załadowanie do rejestru licznika rozkazów odpowiedniego adresu oraz na wyłączenie sygnałem I/OW układu relokacji. Po wykonaniu tej sekwencji wybieranie układu pamięci dokonuje się już normalnie przez układy dekodujące adresy.

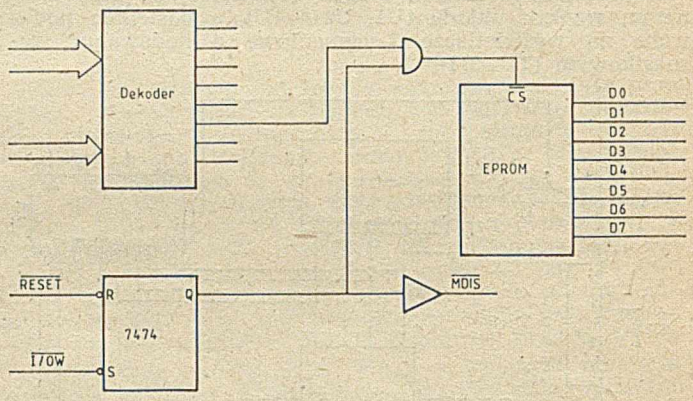
Przerwania wektorowe RST0-RST7 powodują przekazanie sterowania do podprogramu obsługi zaczynającego się od adresu 0, 8, 16, 24, 32, 40, 48, 56. Można uniknąć konieczności rezerwacji początku pamięci dla obsługi przerwań systemowych, wykorzystując w tym celu rozwiązania przedstawione na rysunkach 1 i 3.

Na wejścia zerujące przerzutników należy wprowadzić sygnał będący iloczynem logicznym sygnałów INTA i RESET (rys. 4). W pamięci PROM (64 bajty dla ośmiu przerwań) pod odpowiednimi adresami należy umieścić rozkazy skoków do podprogramów obsługi (rys. 5). Takie rozwiązanie pozwala na umieszczenie programów obsługi przerwań w dowolnym miejscu przestrzeni adresowej, umożliwiając swobodne dysponowanie początkiem pamięci.

MAREK DORYWALSKI
PIE Warszawa

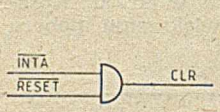
Sposób generowania sygnałów na szynie danych i otrzymywnia konfiguracji bramek

	JMP F800H	D7	D6	D5	D4	D3	D2	D1	Dą	Q1	Q2	Q3	$\overline{Q1}$	$\overline{Q2}$	$\overline{Q3}$
RESET	C3	1	1	0	0	0	0	1	1	0	0	0	1	1	1
PO 1 $\overline{MEMR}$	00	0	0	0	0	0	0	0	0	1	0	0	0	1	1
PO 2 $\overline{MEMR}$	F8	1	1	1	1	1	0	0	0	1	1	0	0	0	1
PO 3 $\overline{MEMR}$	FF	1	1	1	1	1	1	1	1	1	1	1	0	0	0
				$\overline{Q1}+Q2$			Q2				Q3		$\overline{Q1}+Q3$		

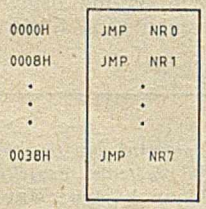


Rys. 3. Układ relokacji

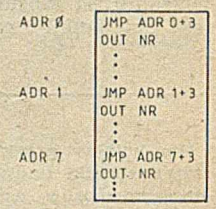
Podobnie można wykorzystać układ relokacji, w którym na wejście R przerzutnika należy wprowadzić iloczyn INTA i RESET. Pierwszymi rozkazami programu obsługi przerwania powinny być JMP i OUT (rys. 6).



Rys. 4. Sygnał zerowania przerzutników



Rys. 5. Obszar adresowy pamięci PROM



Rys. 6. Obszar adresowy podprogramów obsługi przerwań

LITERATURA

[1] Gates replace PROM in Intellec-8 bootstrap loader. Electronics, January 18, 1979, p. 121
[2] PROM adds bootstrap loader to Intellec-8 development system. Electronics, April 27, 1978, p. 126.

Zaczynamy realizować zobowiązania. Oto pierwsze niebanalne SŁOWO. Pozwala ono uporządkować ciąg liczb według wielkości. Zdradzimy jednak, bez przypierania do muru, że jest to nadal bardziej program dydaktyczny niż użytkowy. Musimy jeszcze doskonalić znajomość FORTH'A. Niecierpliwym prymusom proponujemy — jako zadanie domowe — przeprowadzenie optymalizacji. Nadesłane prace zostaną ocenione przez ekspertów mikroKLANU!

FORTH (3)

Programowanie rozpoczniemy od zdefiniowania „własnego" stosu roboczego. Umieścimy go w pamięci powyżej słownika i bufora wejściowego. Wykorzystamy przy tym zmienną **HERE** określającą adres wierzchołka słownika. Określimy także stałą **REZERWA**, która zarezerwuje obszar 1 KB pamięci na bufor wejściowy. Zdefiniujemy również zmienną **HN** zawierającą adres wierzchołka stosu i dwa słowa pomocnicze:

H+ — przesunięcie wierzchołka stosu **H** w górę o 2 bajty i odłożenie aktualnego adresu na stos danych
H— — odłożenie adresu wierzchołka stosu **H** na stos danych i obniżenie wierzchołka stosu **H** o 2 bajty.

Definiowanie stosu (nazwiemy go stosem **H**) opisujemy trzema słowami głównymi:
HSET — inicjowanie stosu **H**
>H — przeniesienie liczby ze stosu danych na stos **H**
H> — przeniesienie liczby ze stosu **H** na stos danych.

Definiowanie stosu H

```
1024 CONSTANT REZERWA (definiowanie stałej REZERWA o wartości 1024)
0 VARIABLE HN (definicja zmiennej HN o wartości początkowej 0)

: HSET HERE @ REZERWA + (obliczenie początku stosu H)
        DUP HN ! (zapisanie w HN)
        0 SWAP ! (zapisanie wysokości stosu H równej 0)
;
: H+ HN@ (adres początku stosu H)
DUP@ 2 + (wysokość stosu H + 2)
OVER OVER SWAP ! (zapis nowej wysokości stosu H)
+ (adres wierzchołka stosu H)
;

      : H— HN@ (adres początku stosu H)
      DUP@ (wysokości stosu H)
       OVER OVER 2 — SWAP ! (zapis wysokości stosu H minus 2)
      +   (adres wierzchołka stosu H)
;
: >H H+ (adres na stosie H)
     !   (zapis)
: H> H— (adres na stosie H)
     @ (pobranie)
;
```

Przykład działania zdefiniowanych słów:
HSET 1 >H 2 >H 3 >H [CR] OK
H>.H>.H>.[CR] 3 2 1 OK

Definiowanie słowa SORT

Możemy teraz przystąpić do właściwego programu sortującego. W tym celu określimy cztery słowa pomocnicze:

MINIMUM — pobiera ze stosu danych liczbę określającą wysokość stosu i odszukuje wartość minimalną wśród liczb zapisanych na stosie; wyniki odkłada na stosie **H: (N1, N2, N3)**
N1 — adres elementu minimalnego na stosie danych (wierzchołek = 1)
N2 — zero
N3 — wartość elementu minimalnego
ZGUB — usuwa ze stosu danych liczbę wybraną przez **MINIMUM**, a ze stosu **H** usuwa liczby **N1** i **N2** zapisane przez **MINIMUM**
ZAPISZ — dokonuje rotacji trzech liczb ze stosu **H** i dekrementuje liczbę znajdującą się na wierzchołku stosu **H** oraz kopiuje wierzchołek stosu **H** na stos danych
ODDAJ — przenosi dane ze stosu **H** na stos danych; liczba z wierzchołka stosu **H** określa liczbę danych do przeniesienia.

```
: MINIMUM
          1 >H >H DUP >H (zainicjowanie wartości N1, N2, N3)
          BEGIN      (otwarcie bloku)
          H> DUP     (aktualna wartość minimalna)
          H> DUP >H 2 + (wysokość badanego stosu plus dwa parametry pobrane z H)
          PICK       (skopiowanie na wierzchołek stosu liczby leżącej na aktualnie „sondowanej" wysokości stosu)
          DUP ROT<   (porównanie z minimum)
          IF         (gdy mniejsze od minimum)
          >H >H >H DROP (zapisanie nowego minimum)
          ELSE       (gdy większe lub równe minimum)
          DROP >H    (zapis starego minimum)
          THEN       (koniec warunku IF)
          H> H>      (pobranie minimum i wysokości stosu)
          1 — SWAP OVER >H >H (zapis wysokości 1 i minimum)
          0 =        (czy wysokość równa 0)
          UNTIL      (powtarzanie bloku BEGIN aż poprzedni warunek będzie spełniony)
;

: ZGUB
          H> H>      (pobranie N1, N2)
          DROP       (zgubienie N2)
          H>         (pobranie N3)
          SWAP >H    (zapis N1)
          ROLL       (rotacja stosu na wysokość N3)
          DROP       (zgubienie elementu wskazanego przez N3)
;

: ZAPISZ
          H> H> H>   (pobranie ze stosu H)
          ROT
          >H >H      (odłożenie na stos H)
          1 —        (zmniejszenie wartości o 1)
          DUP        (kopia dla stosu danych)
          >H         (zapis na stosie H)
;

: ODDAJ
          H>         (pobranie ilości liczb do przeniesienia)
          BEGIN      (otwarcie bloku)
                     H> SWAP (zapis na stosie danych)
                     1 — DUP (zmniejszenie ilości liczb do przeniesienia o 1)
                     0 =     (koniec ? — ilość = 0)
          UNTIL      (powtarzanie bloku BEGIN aż poprzedni warunek będzie spełniony)
          DROP       (zgubienie zbędnego zera)
;

: SORT          (dydaktyczny program sortujący)
      DEPTH (odłożenie na stosie wysokości stosu)
        DUP >H >H (zapis wysokości sortowanego stosu)
```

```
BEGIN (otwarcie bloku)
      H> DUP >H MINIMUM (poszukiwanie minimum)
      ZGUB      (usunięcie znalezionego minimum)
      ZAPISZ    (zapamiętanie znalezionego minimum w odpowiednim miejscu stosu H i zmniejszenie wysokości badanego stosu o 1)
      0=        (wysokość badanego stosu = 0 ?)
      UNTIL     (powtarzanie bloku BEGIN aż poprzedni warunek będzie spełniony)
      H> DROP   (usunięcie zbędnego zera)
      ODDAJ     (przepisanie posortowanego stosu H na stos danych)
```

Przykład użycia programu

```
1 3 5 7 9 2 4 6 8 [CR] OK
HSET [CR] OK
SORT [CR] OK
..... [CR] 1 2 3 4 5 6 7 8 9 OK
```

Opisany program sortujący można łatwo zoptymalizować. Spróbuj!

MAREK CZARZYŃSKI
ABAKUS — Warszawa

TEST

Mikrokomputery bardzo trudno porównywać. Każdy ma swoje niejako dodatkowe możliwości, a także skrzętnie ukryte niedomagania. Pewnym ułatwieniem jest fakt, że prawie każdy może realizować programy napisane w języku BASIC. Ale — żeby życie nie było zbyt łatwe — nie ma chyba dwóch identycznych adaptacji tego języka. Projektanci lekką ręką dodają rozszerzenia i równie lekko wprowadzają ograniczenia. Rozpowszechniane są jednak programy wykorzystujące rdzeń języka, które pozwalają ocenić szybkość realizacji pewnych najczęściej stosowanych procedur. Programy takie (ang. Benchmark Program) często są pisane na zamówienie producentów, tak aby wykazać wyższość konkretnego komputera nad innymi. Zamieszczone poniżej procedury wydają się być stosunkowo obiektywne. Zaczerpnięte zostały z biuletynu Personal Programing Center — PPC Computer Journal V1N1P10 1982. Poniżej podajemy tabelę z wynikami testu dla siedmiu popularnych mikrokomputerów. Jeżeli nie ma wśród nich Twojego, będziemy wdzięczni za przesłanie nam uzupełnienia do tabeli.

Program 1

```
300 PRINT "START"
400 FOR K=1 TO 1000
500 NEXT K
700 PRINT "END"
800 END
```

Program 2

```
300 PRINT "START"
400 K=0
500 K=K+1
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

Program 3

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/K*K+K—K
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

Program 4

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/2*3+4—5
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
```

Program 5

```
300 PRINT "START"
400 K=0
500 K=K+1
510 A=K/2*3+4—5
520 GOSUB 820
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

Program 6

```
300 PRINT "START"
400 K=0
430 DIM M(5)
500 K=K+1
510 A=K/2*3+4—5
520 GOSUB 820
530 FOR L=1 TO 5
540 NEXT L
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

Program 7

```
300 PRINT "START"
400 K=0
430 DIM M(5)
500 K=K+1
510 A=K/2*3+4—5
520 GOSUB 820
530 FOR L=1 TO 5
535 M(L)=A
540 NEXT L
600 IF K<1000 THEN 500
700 PRINT "END"
800 END
820 RETURN
```

Oprac.
J. TATARKIEWICZ

Tabela wyników testu

Komputer	Język	Czas wykonywania poszczególnych programów (s)							
		1	2	3	4	5	6	7	suma
OSBORNE I	MBASIC	1,5	4,5	12,1	11,8	12,8	23,0	36,2	101,9
APPLE II	APPLESOFT	1,4	8,3	15,7	17,5	19,1	28,4	44,5	134,9
HP-85	HP BASIC	2,5	4,5	17,1	17,2	18,4	30,4	45,5	135,6
HP-75	HP BASIC*)	3,1	5,1	21,6	21,3	26,3	41,9	59,4	178,7
ZX SPECTRUM	BASIC	4,3	8,14	20,02	19,26	23,06	53,12	77,48	205,42
TRS-80	BASIC II	2,8	11,3	27,7	29,5	32,8	54,5	82,5	241,1
SHARP PC-1500	BASIC II.	14,8	29,5	78,9	79,5	94,0	168,9	223,4	689,0
Twój komputer	BASIC	...	...	...	...	...	...	...	...

*) zmienne typu REAL

Potrzebna pomoc

(pod tym hasłem zamieszczać będziemy anonse firm i osób prywatnych)

- Firma INSTAL zakupiła mikrokomputery IMP85. Poszukuje oprogramowania użytkowego pomocnego w prowadzeniu przedsiębiorstwa. Poszukiwana jest również realizacja podłączenia IMP85 do ODRY 1305 (pracującej pod systemem GEORGE-3) w charakterze inteligentnego terminala, mogącego wyeliminować dziurkarki kart. Zgłoszenia przyjmuje Jarosław Zrost; INSTAL—Łódź, ul. Brukowa 20, tel. 51-21-19 w. 285.
- Otrzymaliśmy informację od Jana Rabizo z Sosnowca o istnieniu Agencji Mikro-Komputerowej AMiCO, oferującej usługi obliczeniowe, programy mikrokomputerowe i doradztwo informatyczne. Programy można otrzymać w postaci wydruku lub zapisane na taśmie magnetycznej.
  Kontakt: Agencja AMICO, Sosnowiec, P-157, tel. 69-96-49 (godz. 18—20).
- Pisaliśmy już o możliwości wymieniania programów dla ZX SPECTRUM. Zbiór Piotra Parlewicza (tel. 42-40-94 w Warszawie) przekroczył liczbę 100!

- **INFORMATYKA tel. 27-71-40**
- **ABAKUS tel. 42-91-85**
- **KUM tel. 41-26-01**
- **?**

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

RYSZARD GACEK
Instytut Elektroenergetyki
Akademia Górniczo-Hutnicza
Kraków

ANDRZEJ GOŚCIŃSKI
Instytut Informatyki
Akademia Górniczo-Hutnicza
Kraków

ZBIGNIEW REMPAŁA
Zakłady Azotowe
Tarnów

ROMAN RYGAŁ
Instytut Elektroenergetyki
Politechnika Częstochowska

Analizator ATA-77 w systemie CAMAC do analizy dwuparametrycznej

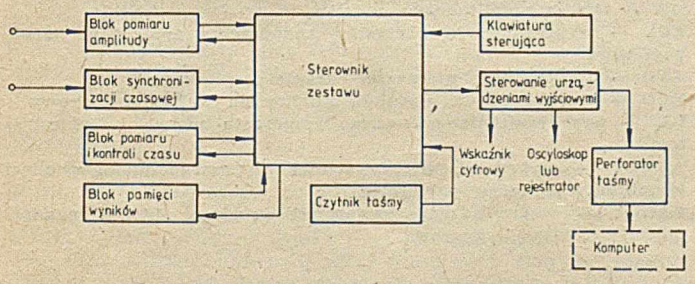
Dwuparametryczny analizator amplitudowo-czasowy zaprojektowany i zrealizowany w systemie CAMAC, służy do obliczania energii wyładowań w dielektrykach na podstawie bezpośrednich pomiarów [1, 2]:
— amplitudy pojedynczego impulsu związanego z wyładowaniem
— wartości chwilowej napięcia zasilającego w chwili wystąpienia wyładowania.

Ze względu na wymagania użytkowe system powinien spełniać następujące warunki:
— duża uniwersalność
— łatwa adaptacja do potrzeb planowanych eksperymentów
— łatwość obsługi
— zobrazowanie wyników w celu doboru warunków eksperymentu
— forma otrzymanych wyników dostosowana do dalszego opracowania przy użyciu komputera
— minimalny czas martwy, tj. dopuszczalny czas między dwoma kolejnymi impulsami.

Wymienione wymagania zrealizowane przez dobór odpowiednich bloków systemu CAMAC, typowych urządzeń wejścia-wyjścia, odpowiednią konstrukcję programów i jednolitość poleceń operatora we wszystkich programach użytkowych.

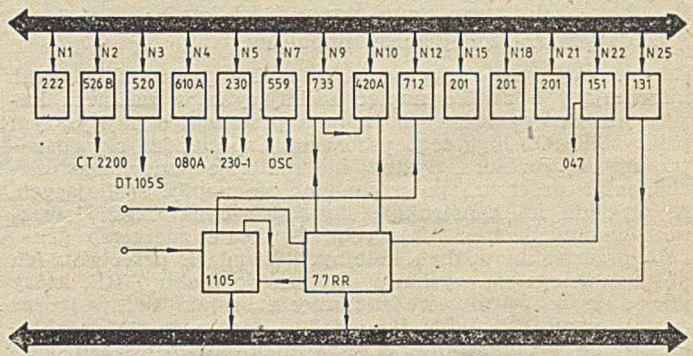
KONFIGURACJA SYSTEMU ATA-77

W systemie ATA-77 można wyróżnić sześć podstawowych modułów funkcjonalnych, które realizują zadania wynikające z postawionych wymagań (rys. 1). Sterownik zestawu kontroluje pracę wszystkich bloków CAMAC wchodzących w skład systemu, przez wykonywanie odpowiednich procedur przechowywanych w pamięci ferrytowej. Komunikacja operatora z systemem odbywa się za pośrednictwem klawiatury sterującej. Blok pomiaru i kontroli czasu umożliwia odmierzanie czasu trwania analizy oraz pomiar czasu nadejścia impulsu. Wyniki pomiarów są przechowywane w pamięci ferrytowej. W celu zsychronizowania czasu nadejścia impulsu z napięciem wymuszającym wyładowania niezupełne, konieczne było wykonanie niestandardowego bloku 77RR [3].



Rys. 1. Schemat funkcjonalny analizatora amplitudowo-czasowego ATA-77

W systemie ATA-77 jako jednostkę przetwarzającą i sterującą zastosowano procesor autonomiczny typu 131 (rys. 2). Wraz z odpowiednim programem zapewnia on minimalny czas martwy podczas analizy (maksymalny czas martwy — 35 μs). W przypadku zastosowania komputera, np. MERA-400, do sterowania tym systemem czas martwy wydłużył się kilkakrotnie i wynosił ok. 100 μs.



Rys. 2. Architektura analizatora w systemie CAMAC (222 — pamięć stała, 526B — sprzęg czytnika, 520 — sprzęg dziurkarki, 610A — konwerter kodu, 080A — wskaźnik cyfrowy, 230 — pamięć przełącznikowa, 559 — blok sterowania monitorem, 733 — czasomierz, 420A — licznik dwójkowy, 712 — przetwornik a/c, 201 — pamięć ferrytowa 1K, 151 — blok przerwań, 131 — procesor autonomiczny, 1105 — bramka liniowa, 77RR — blok synchronizacji)

Poza wykonywaniem procedur sterujących, procesor przyjmuje sygnały przerwań z bloku typu 151 oraz generuje sygnały sterujące dla bloku synchronizacji czasowej. W torze pomiaru amplitudy impulsu wykorzystano spektrometryczny przetwornik analogowo-cyfrowy typu 712 (lub 711). Pomiar czasu trwania analizy jest dokonywany przy użyciu czasomierza typu 733, który jednocześnie służy jako generator impulsów zegarowych. Dzięki zliczaniu tych impulsów przez jeden z liczników bloku typu 420A możliwy jest pomiar czasu nadejścia impulsu analizowanego. Drugi licznik tego bloku umożliwia kontrolę liczby impulsów przychodzących z procesu. Wyniki pomiarów są przechowywane w dwóch blokach pamięci ferrytowej typu 201.

Zespół układów sterujących urządzeniami wyjściowymi umożliwia zobrazowanie i utrwalenie wyników analizy. W charakterze układów sterujących wykorzystano: przetwornik cyfrowo-analogowy typu 559 (do sterowania monitorem ekranowym, oscyloskopem oraz rejestratorem), konwerter typu 610A kodu dwójkowego na kod BCD (do sterowania wskaźnikiem cyfrowym), blok sprzęgający perforatora taśmy papierowej typu 520 (do sterowania dziurkarką DT 105-S). Do wprowadzania oprogramowania służy czytnik taśmy papierowej CT2200 sterowany programem umieszczonym w bloku pamięci stałej typu 222.

Powyższe bloki umieszczono w kasecie CAMAC, z której są zasilane i sterowane za pośrednictwem magistrali. W drugiej kasecie umieszczono bloki nie sterowane przez magistralę, tj. bramkę liniową typu 1105 oraz niestandardowy blok 77RR, tworzące układ synchronizacji czasowej.

DZIAŁANIE ZESTAWU

W działaniu zestawu należy rozróżnić proces wykonywania pomiaru i zobrazowanie wyników.

Współpraca bloków CAMAC podczas analizy

W ramach przygotowania systemu do analizy amplitudowo-czasowej procesor wpisuje — z tablicy przełącznikowej zaprogramowanej przez operatora — wartości początkowe do liczników bloku 420A, czas trwania analizy — do rejestru bloku 733 oraz ustala częstotliwość impulsów zegarowych generowanych przez ten blok. Następnie procesor wysyła impuls do bloku 77RR, oznaczający zezwolenie na otwarcie bramki i przechodzi do stanu oczekiwania na zgłoszenie lub przerwanie. Równocześnie z otwarciem bramki liniowej przez impuls z detektora generowany jest impuls inicjujący odmierzanie czasu przez blok typu 733 oraz otwierane są liczniki bloku 420A. Po przejściu przez bramkę liniową impuls pomiarowy podlega przetworzeniu na sygnał cyfrowy. Po zakończeniu konwersji przetwornik wysyła zgłoszenie L do procesora, który odczytuje wynik i umieszcza go w pamięci ferrytowej. Następnie procesor wpisuje do pamięci ferrytowej aktualny stan licznika impulsów zegarowych i przechodzi w stan PAUSE. Opisany cykl jest powtarzany dla każdego impulsu pochodzącego z procesu.

Zakończenie analizy może nastąpić na skutek jednego z trzech zdarzeń:

- zaniku napięcia zasilającego
- upływu założonego czasu trwania analizy
- nadejścia założonej liczby impulsów.

Zdarzenia te są obsługiwane w trybie przerwań. Sygnał informujący o zaniku napięć zasilających w zasilaczu kasety jest doprowadzony do kanału przerwań o najwyższym priorytecie. Obsługa tego przerwania sprowadza się do zatrzymania procesora. System nie realizuje ponownego uruchomienia procesora po pojawieniu się napięć zasilających, ze względu na konstrukcję stanowisk badawczych (styczniki w obwodach zasilających). Pozostałe zdarzenia generują przerwania w dwu kolejnych kanałach przerwań. Ich obsługa polega na wysłaniu impulsu do bloku 77RR, który powoduje bezwarunkowe zamknięcie bramki liniowej oraz zablokowanie liczników typu 420A. Następuje programowe kasowanie i blokowanie przetwornika analogowo-cyfrowego, czasomierza typu 733 oraz liczników bloku 420A.

Analiza amplitudowa jest realizowana analogicznie jak amplitudowo-czasowa, przy czym sygnał zgłoszenia z przetwornika analogowo-cyfrowego powoduje akcję procesora polegającą na odczytaniu, skasowaniu i odblokowaniu przetwornika oraz dodaniu jedynki do słowa pamięci, wyznaczonego przez amplitudę analizowanego impulsu. Zakończenie analizy amplitudowej następuje wskutek upływu założonego czasu jej trwania lub zaniku napięcia zasilającego.

Współpraca bloków CAMAC podczas wyprowadzania wyników analizy

Procesor przesyła do rejestru wejściowego bloku 559 zawartość kolejnych słów wybranego bloku pamięci (wynik pomiaru amplitudy lub czasu), wskutek czego następuje rysowanie na ekranie oscyloskopu odpowiedniego widma — amplitudowego lub czasowego. Fragment widma, odpowiadający kanałowi wskazanemu przez operatora, jest dodatkowo rozjaśniany, co umożliwia łatwą identyfikację punktów charakterystycznych.

W przypadku rysowania widma na rejestratorze zawartość pamięci jest przesyłana analogicznie jak przy współpracy z oscyloskopem. Szybkość przesyłania kolejnych słów jest dostosowana do możliwości użytego rejestratora samopiszącego.

Zawartość słowa pamięci, wskazanego przez operatora, jest wyświetlana także na wskaźniku cyfrowym typu 080 po uprzedniej konwersji z kodu dwójkowego na BCD w bloku 610A. Jednocześnie z zawartością słowa wyświetlany jest jego adres (numer kanału lub impulsu). Istnieje także możliwość automatycznego wyświetlania kolejnych słów pamięci z dowolną szybkością.

Perforowanie wyników analizy na taśmie papierowej następuje przy użyciu dziurkarki DT105S. Procesor dokonuje konwersji kolejnych słów wybranego obszaru pamięci z kodu dwójkowego na kod BCD, a następnie wysyła je do rejestru wejściowego bloku sprzęgającego perforatora. Transmisja jest synchronizowana sygnałami zajętości dziurkarki.

OPROGRAMOWANIE SYSTEMU ATA-77

Oprogramowanie systemu składa się z dwóch części:
— programów systemowych (namiastka systemu operacyjnego), inicjujących i kontrolujących pracę zestawu
— programów użytkowych realizujących żądane funkcje.

Rozdzielenie programów systemowych od użytkowych jest podyktowane stworzeniem takiego systemu, który przy niewielkich zmianach umożliwia realizację nowych funkcji. Oprogramowanie zostało wykonane bezpośrednio w języku wewnętrznym procesora typu 131. Duży wpływ na możliwości programowe systemu miała mała pojemność bloku pamięci programowej (1 K słów).

Programy systemowe

Programy systemowe zapewniają przygotowanie zestawu do pracy przez ustawienie czasu trwania analizy, wartości początkowych liczników i czasomierza, wybór jednostki pomiarowej czasu oraz liczby kanałów. Umożliwiają uruchomienie i zatrzymanie zestawu oraz różny rodzaj pracy (po wykonaniu każdego programu użytkowego konieczna jest interwencja operatora). Inicjowanie wykonania programu systemowego jest dokonywane przez wciśnięcie odpowiedniego przycisku tablicy przełącznikowej.

System ATA-77 realizuje następujące polecenia:

NEW — zerowanie systemu, wpisanie parametrów
TIME — ustawianie systemowego zegara czasu rzeczywistego
SMBL — wpis symbolu identyfikującego wykonywane pomiary
START — zapoczątkowanie wykonywania programów użytkowych
STOP — przerwanie realizacji programów użytkowych
CONT — żądanie przejścia do realizacji następnego programu użytkowego (przy pracy w trybie pojedynczego programu lub eksperymentu [1]).
RET — STOP w trybie pojedynczego programu lub eksperymentu
NULL — przejście do realizacji następnego eksperymentu
USW, REP — symulacja poleceń NEW i START, która umożliwia automatyczne wykonywanie programów
STEP — tryb pojedynczego programu, gdy po wykonaniu dowolnego programu użytkowego system oczekuje na decyzję operatora (CONT lub RET)
HLT — tryb pojedynczego eksperymentu, gdy po zrealizowaniu eksperymentu system oczekuje na decyzję operatora
END — po wykonaniu danego eksperymentu przejście do stanu oczekiwania na polecenie NEW.
Polecenia NULL, USW, REP, STEP, HLT, END są indywidualne dla każdego eksperymentu.

Programy użytkowe

Mała pojemność pamięci oraz brak monitora znakowego powodują, że jedynym możliwym rozwiązaniem dotyczącym aktywizacji programów użytkowych w ramach eksperymentu jest sztywna lista programowa, której kolejność ustala programista. Dobór tych programów jest zdeterminowany przeznaczeniem systemu. Analizator ATA-77 wyposażono w następujące programy:

LFI — korekcja symbolu identyfikacyjnego
LOG — wyprowadzenie opisu eksperymentu na taśmę papierową
ATA — analiza amplitudowo-czasowa
AA — analiza amplitudowa
EOE — wyprowadzenie warunków zakończenia analizy na taśmę papierową
TVD — zobrazowanie widma na ekranie oscyloskopu
BCD — wyprowadzenie wyników analizy na wskaźnik cyfrowy
GP — wyprowadzenie wybranego fragmentu widma na rejestrator
AAOUT — wyprowadzenie zawartości pierwszego bloku pamięci (amplitudy) na taśmę papierową
TAOUT — wyprowadzenie zawartości drugiego bloku pamięci (czasu) na taśmę papierową.

---

[1]) Eksperymentem nazywamy ciąg programów użytkowych, które są aktywizowane dla jednego kompletu danych wejściowych

Wymienione programy użytkowe mogą być aktywizowane w wymienionej kolejności, jednokrotnie w ramach każdego z eksperymentów, z parametrami określającymi: czas trwania analizy, liczbę impulsów przyjmowanych w analizie amplitudowo-czasowej lub liczbę kanałów dla analizy amplitudowej oraz żądaną dokładność pomiaru czasu. Parametry, indywidualne dla każdego eksperymentu, są przekazywane za pośrednictwem tablicy przełącznikowej.

* * *

Przedstawiony system ATA-77 analizuje impulsy o czasie narastania 0,2—40 μs, czasie opadania 0,5 μs oraz amplitudzie 50 mV—10,24 V.

Przedstawiony system ATA-77 analizuje impulsy o czanych kompletów danych wejściowych, a w ramach każdego eksperymentu można akytwizować osiemnaście programów użytkowych.

Obsługa programów użytkowych może następować w czterech trybach: pojedynczego programu, pojedynczego eksperymentu, do polecenia END, w cyklicznie powtarzanej sekwencji eksperymentów.

Rozwiązaniem rozszerzającym możliwości pomiarowe systemu jest zastosowanie niestandardowego bloku 77RR do synchronizacji czasowej. Dzięki jego konstrukcji, analizę procesu można wykonywać w sposób ciągły lub w ściśle zdefiniowanych momentach (np. istnieje możliwość uzykania tzw. okna czasowego wyzwalanego napięciem wymuszającym wyładowanie niezupełne).

Wadą systemu jest konieczność zaplanowania wszystkich aktywizowanych eksperymentów i podania danych wejściowych przed ich uruchomieniem (tj. przed wydaniem polecenia START) oraz — niewielka liczba analizowanych impulsów w przypadku analizy amplitudowo-czasowej. Przedstawiany system, oprócz zastosowań w badaniach zjawiska wyładowań niezupełnych, może być użyty w dowolnych pomiarach widma amplitudowego lub amplitudowo-czasowego. Osiągnięte wyniki stanowią kres możliwości dla tego zestawu sprzętowego.

Analizator ATA-77 stanowi kontynuację prac nad zastosowaniem systemu do automatyzacji eksperymentów w laboratorium wysokich napięć i obecnie jest efektywnie wykorzystywany do prowadzenia badań. Rozpoczęto działania zmierzające do takiego rozszerzenia możliwości analizatora, aby można prowadzić eksperymenty sterowane komputerowo.

LITERATURA

[1] Gacek R., Gościński A., Mordarski G.: Wielokanałowy analizator amplitudy na bazie systemu CAMAC do rejestracji impulsów wyładowań niezupełnych. Kwartalnik AGH „Elektrotechnika" (w druku)

[2] Rempała Z., Rygał R.: Dwuparametryczny analizator wyładowań niezupełnych na bazie systemu CAMAC. Praca dyplomowa, Instytut Informatyki, Akademia Górniczo-Hutnicza, Kraków 1982

[3] Rempała Z., Rygał R.: Niestandardowy blok 77RR systemu CAMAC do synchronizacji czasowej. Biuletyn Techniczno-Informacyjny MERA, nr 11 (245), str. 26, 1982.

Przegląd języków wysokiego poziomu (5)

Rozwój i tematy prac badawczych

Specjaliści od języków programowania prowadzą obecnie prace badawcze dotyczące różnych problemów z tego działu informatyki. Niektórymi tematami (np. automatycznym programowaniem) zajmują się od dawna; na inne (np. mierzalność języków) dopiero zaczynają zwracać uwagę. Ze względu na ograniczenia rozmiarów artykułu można w nim zwrócić uwagę tylko na niektóre zagadnienia, takie jak: **ogólność danych** (ang. data abstraction), **weryfikacja** (ang. verification), **niezawodność** (ang. reliability), **automatyczne programowanie** (ang. automatic programming), **języki funkcjonalne** (ang. functional languages), **języki zapytań** (ang. query languages) oraz **mierzalność języków** (ang. language metrics).

OGÓLNOŚĆ DANYCH

W drugiej połowie lat siedemdziesiątych terminu **ogólność danych** zaczęto używać coraz częściej. Pojęciem **ogólności** posługiwano się od dłuższego czasu do określania takich elementów, jak procedura, podprogram czy funkcja. Konstrukcje te zawierają kod realizujący zadane czynności. Wykonywane są na zlecenie użytkownika, który nie wie i nie interesuje się sposobem uzyskania końcowego wyniku. Trywialnym przykładem może być elementarna funkcja matematyczna, taka jak pierwiastek kwadratowy czy sinus. Bardziej złożone procedury mogą zawierać sortowanie, a nawet duże fragmenty programu. Dlatego też mówi się, że każdy program jest tylko procedurą innego programu.

Główna cecha ogólności to możliwość wykorzystywania, przez użytkownika piszącego program, wyników otrzymywanych przez inny program: użytkownik ten nie musi wiedzieć jak program działa i — co ważniejsze — nie ma nad nim żadnej kontroli. Przy ogólności danych format danych jest przed programistą-użytkownikiem utajony, a z operacji na danych znane mu są tylko te, które może sam wykonywać.

Język umożliwiający realizację koncepcji ogólności danych powinien:

- zawierać konstrukcję pozwalającą na stosowanie ogólności danych jako odrębnej jednostki; korzystanie z tej konstrukcji powoduje wybranie reprezentacji obiektów danych oraz zdefiniowanie algorytmów wszystkich operacji, przy użyciu terminów tej reprezentacji
- ograniczyć dostęp do tej reprezentacji, tzn. dopuścić tylko operacje na reprezentacji, które całkowicie charakteryzują zachowanie się obiektów.

Najprostszym przykładem ogólności danych jest stos, pozwalający na wkładanie i usuwanie obiektów oraz tworzenie i testowanie stosu — operacji mniej używanych, choć równie koniecznych. Najważniejsze są tu: typ danej oraz operacje związane z daną tylko określonego typu.

W procedurze podaje się pewną liczbę parametrów wejściowych, natomiast wynikiem są parametry wyjściowe. Z kolei w COBOLU dane są dostępne dla wszystkich programów.

Wstęp do koncepcji ogólności danych został podany przez Shankera, a opis problemów językowych i ogólności — przez Shawa (1980).

WERYFIKACJA

Prace naukowe związane z weryfikacją programów rozwijają się począwszy od wczesnych lat siedemdziesiątych. O rozwoju tej dziedziny świadczy fakt, że w książce autorki z 1969 roku nie było jeszcze żadnych wzmianek na ten temat, choć przytoczona bibliografia liczyła już ponad 800 pozycji. W tamtych czasach ukazywało się niewiele publikacji w ramach tej dziedziny; obecnie można już stwierdzić, że prowadzi się poważne prace badawcze na ten temat, a większość z nich bezpośrednio lub pośrednio doty-

czy języków programowania. Celem weryfikacji jest pokazanie lub dowiedzenie, że program komputerowy jest zgodny ze szczegółowym opisem jego działania (London, 1979).

Istotny jest zwrot „zgodny ze szczegółowym opisem", gdyż zarówno w chwili obecnej, jak i w najbliższej przyszłości program można porównywać wyłącznie z jego opisem. Opis ten niekoniecznie musi być reprezentacją rzeczywistych wymagań stawianych programowi. Weryfikacja prowadzi do problemów związanych z językami, gdyż opis programu musi być przedstawiony w pewnego rodzaju języku, a weryfikowany program jest zwykle napisany w języku wysokiego poziomu. Metoda asercji (ang. assertions), wprowadzona w 1967 roku przez Floyda, należy do podstawowych w tej dziedzinie. Polega ona na tym, że w odpowiednich miejscach programu programista wstawia asercje dotyczące określonych zmiennych. Dowód polega na sprawdzeniu, że każda z asercji (również umieszczonych w pętlach) jest zawsze prawdziwa.

W początku lat osiemdziesiątych zweryfikowano, używając różnych metod, pewną liczbę stosunkowo małych programów. Niestety, dowody były znacznie dłuższe od tych programów.

Czynione są próby zdefiniowania języków szczególnie nadających się do weryfikacji. Najbardziej znanym z nich jest EUCLID.

NIEZAWODNOŚĆ

Problemy dotyczące związków niezawodnego oprogramowania z językami programowania omawiano już w 1977 roku na specjalnej konferencji ACM. W początkach rozwoju informatyki główny nacisk kładziono na wydajność i przenośność oprogramowania. Z biegiem lat jednak technika komputerowa przenika do coraz większej liczby dziedzin naszego życia i jej niezawodność nabiera ogromnego znaczenia. Zdaniem autorki, w przyszłości niezawodność stanie się prawdopodobnie podstawowym kryterium oceny, ważniejszym niż wydajność programu. Niezawodność sprzętu zwiększa się coraz szybciej, natomiast oprogramowania tylko w nieznacznym stopniu. Odpowiedź na pytanie, które cechy języka pomagają w tworzeniu niezawodnego oprogramowania nie może być według autorki, obiektywna.

Nie istnieje dotąd ścisła definicja cech, które pomagają w tworzeniu niezawodnego oprogramowania. Wszyscy zgadzają się natomiast, że „konieczność określania typów" jest cechą ważną. Oznacza to takie mechanizmy w języku, dzięki którym każda zmienna musi być zadeklarowana jako zmienna określonego typu lub postaci, natomiast zmienne różnych typów mogą być łączone dopiero po wywołaniu odpowiednich procedur konwersji; łamanie tych reguł powinno być wykrywane przez kompilator. Okazuje się, że nawet tak „archaiczne" języki, jak FORTRAN, miały duże wymagania dotyczące typów: przynajmniej pod jednym względem — nie wolno było dodawać liczb całkowitych do zmiennoprzecinkowych. Spośród późniejszych języków krańcowo słabe określanie typów występuje w PL/I, w którym można wykonywać działania na prawie dowolnego typu danych. W podręcznikach PL/I zasady wykonywania takich działań zajmują wiele stron. Nowsze języki, takie jak PASCAL czy ADA, mają silniejsze określanie typów, choć nie wszyscy są zgodni, że zwiększa to niezawodność programu. Możliwość kombinowania danych różnych typów pozwala na pisanie krótszych programów, a wiemy, że niezawodność można osiągnąć tym łatwiej im program jest krótszy.

Na wspomnianej już konferencji ACM właściwie wszystkie wystąpienia dotyczyły różnych języków programowania i przedstawiały powody, dla których autorzy referatów uważali te języki za bardziej niezawodne. Natomiast prawie nie dyskutowano i nie ustalono, co oznacza pojęcie **niezawodne oprogramowanie**, ani też jak języki programowania mają ten problem rozwiązać.

AUTOMATYCZNE PROGRAMOWANIE

W pierwszych latach istnienia komputerów termin **automatyczne programowanie** oznaczał wyłącznie narzędzie lub technikę łatwiejszą dla programisty niż kodowanie ósemkowe, dwójkowe lub w adresach rzeczywistych.

Termin ten stał się wkrótce synonimem języków wysokiego poziomu, a następnie został zastąpiony przez **automatyczne kodowanie** (ang. automatic coding), lepiej określające pomoc, jaką dawały języki wysokiego poziomu i ich translatory. Obecnie powrócono do punktu wyjścia, gdyż **automatyczne programowanie** ma potocznie dwa znaczenia — jedno odnosi się do języków bardzo wysokiego poziomu (ang. very high level), czyli nieproceduralnych (ang. nonprocedural), drugie zaś — do prób określenia co zamiast jak należy zrobić.

Rozważając szeroki zakres problemów związanych z programowaniem należy zauważyć, że oprócz języków istnieją inne aspekty automatyzacji. Przykładowo — systemy operacyjne i systemy zarządzania bazami danych pomagają w automatycznym generowaniu rozwiązań problemu. Koncepcje wspomagania organizacyjnego programowania czy systemów programowania bardzo wysokiego poziomu dostarczają narzędzi ułatwiających pracę programisty i — wobec tego — znacznie wychodzą poza problemy językowe.

Z punktu widzenia języków **automatyczne programowanie** oznacza w 1980 roku, używanie języków wyższego poziomu niż COBOL, FORTRAN czy PASCAL. Ocena, które języki są bardzo wysokieko poziomu (bardziej nieproceduralne), może być — z braku ścisłych kryteriów — tylko intuicyjna. Jako przykład języka bardzo wysokiego poziomu często podawany jest SETL.

Automatyczne programowanie uważane jest często za dziedzinę sztucznej inteligencji, gdyż ma ono pozwolić użytkownikowi na określanie żądanego wyniku, a nie sposobu jego uzyskania.

Należy wspomnieć o systemach automatycznego programowania, zawierających podstawową wiedzę z określonych dziedzin nauki, jak np. praktycznie już wykorzystywane MYCIN i DENDRAL. MYCIN jest systemem wspomagającym wydawanie diagnoz i określanie leczenia chorób spowodowanych zakażeniami krwi, natomiast DENDRAL realizuje analizę spektrograficzną.

Prace nad tego typu sytemami są kontynuowane, lecz we wszystkich przypadkach okazały się mało wydajne. Czas wykonania niezoptymalizowanego programu napisanego w jednym z takich języków, nawet na bardzo szybkim komputerze jest niemożliwy do przyjęcia. Nie opracowano również odpowiednich metod optymalizacji języków do automatycznego programowania.

PROGRAMOWANIE FUNKCJONALNE

Artykuł przeglądowy, poruszający tak wiele zagadnień, nie spełniłby swego zadania, gdyby nie wspomniano w nim o prowadzonych w ostatnich latach pracach badawczych Johna Backusa. W wystąpieniu związanym z przyznaniem mu w 1978 roku przez ACM nagrody Turinga, uczony ten przedstawił główne wady koncepcji współczesnych języków programowania. Ich przyczyną jest tzw. zator von Neumanna (ang. von Neumann bottleneck), wynikający z cechy sprzętu, która wymaga przesyłania między jednostką centralną a pamięcią pojedynczych słów. Backus wskazał także, że obecnie istniejące języki programowania zawierają:

— zmienne symulujące pamięć maszyny

— instrukcje sterujące, odpowiadające instrukcjom testującym i skokom

— instrukcje podstawiania, odpowiadające pobieraniu, przechowywaniu i wykonywaniu operacji arytmetycznych; instrukcja taka jest dla języków programowania zatorem von Neumanna, zmuszającym do myślenia w kategorii pojedynczych słów.

Backus opisuje klasę prostych systemów programistycznych, które nazywa **systemami programowania funkcjonalnego** (ang. Functional Programming — FP), w których programy są po prostu funkcjami bez zmiennych. Opierając się na tym pojęciu definiuje on tzw. **systemy formalne programowania funkcjonalnego** (ang. Formal Systems for Functional Programming — FFP), o których stwierdza, że „są prostsze od przedstawionych wcześniej języków redukcyjnych, choć są do nich podobne".

System FFP jest zbudowany z cząsteczek, których następnie używa się do budowania **obiektów** i **wyrażeń**. **Aplikacja** jest szczególną postacią wyrażenia (**x : y**), w którym **x** i **y** są także wyrażeniami. Ponadto istnieją tam takie elementy, jak **sprowadzanie**, **przechowywanie** (obydwa odnoszą się do szczególnych obiektów zwanych komórkami) oraz **metakompozycja**.

Backus wprowadza także koncepcję **aplikacyjnego systemu przekształcania stanów** (ang. Applicative State Transition System — AST), w którym widzi alternatywę dla systemu von Neumanna, ukształtowanego przez „normalne" języki (np. ALGOL). System AST jest zbudowany z podsystemów (takich jak system FFP), definicji i zbioru reguł określających sposób przetwarzania danych wejściowych na wyjściowe.

Niektóre elementy opisane przez Backusa mogą przypominać charakterystykę języków LISP i APL, w rzeczywistości jednak stworzył on zupełnie nową koncepcję języków programowania. Na razie specjaliści od programowania funkcjonalnego (np. Landin, 1964; Burge, 1975) nie potrafili zmienić podstawowej struktury obecnie stosowanych języków programowania. Przyszłość pokaże, czy koncepcje Backusa zostaną urzeczywistnione.

JĘZYKI ZAPYTAŃ

Zapotrzebowanie na języki zapytań rozwija się wraz ze wzrostem znaczenia baz danych oraz systemów zarządzania bazami. Przykładowo — administrator bazy danych musi dysponować językiem pozwalającym na jej tworzenie i aktualizację. Z drugiej strony — istnieją ludzie zainteresowani przede wszystkim opisem fizycznej struktury danych, co prowadzi do języka opisu danych (ang. Data Description Language — DDL) i problemu przenośności bazy w przypadku zmiany sprzętu. Użytkownik bazy natomiast wyszukuje informacje nie troszcząc się o ich wewnętrzną organizację.

Języki przeznaczone dla użytkowników można podzielić na niezależne oraz zanurzone w innym języku. W pierwszym przypadku użytkownik musi znać tylko określony język zapytań, w drugim — instrukcje wyszukiwawcze są „dołączone" do języka-gospodarza (ang. host language), takiego jak PL/I lub COBOL.

Z językami niezależnymi związany jest problem ich naturalności. Na przykład — zapytanie „podaj dostawców, którzy dostarczają części wkyorzystywane przez dział 50" powinno być sformatowane w sposób wymagany przez dany system, choć najlepiej byłoby podać je w postaci naturalnej. Z zagadnieniem tym są związane języki „dwuwymiarowe" (ang. two-dimensional languages), tzn. języki oparte na słowach kluczowych (ang. key-word) lub języki o stałym formacie (ang. fixed format). Ważnym problemem jest tu także dozwolony stopień konwersacji między użytkownikiem a maszyną, gdyż w trybie czysto wsadowym użytkownik musi zwracać większą uwagę na sposób formułowania zapytania. Najlepszy — według autorki — przegląd tych języków znajduje się w raporcie Lehmanna i Blasera (1979).

MIERZALNOŚĆ JĘZYKÓW

Nie znany jest dotąd żaden program badań dotyczących mierzalności języków (ang. language metrics), jest to jednak dziedzina, którą należałoby szybko rozwinąć. Należy zauważyć, że **mierzalność języków** i **mierzalność programów** (ang. program metrics) to zupełnie różne pojęcia. Mierzalność programów jest już tematem wielu prac badawczych. Program można oceniać pod względem długości, liczby zmiennych, czasu potrzebnego do napisania, czasu wykonywania, długości kodu wynikowego, liczby błędów wykrytych po uruchomieniu programu itp. Niektóre z tych cech, np. czas wykonywania, zależą także od kompilatora oraz sprzętu, na którym program jest uruchamiany i wykonywany.

Cechy, według których ocenia się język programowania wysokiego poziomu są zupełnie inne. Mogą to być np.:

- elementy związane z projektem języka, takie jak liczba słów kluczowych czy liczba reguł wnioskowania w gramatyce
- przydatność języka dla określonej dziedziny zastosowań; w jaki sposób można jednak stwierdzić, że np. język COGO bardziej nadaje się do budownictwa, a GPSS do symulacji dyskretnej, a nie odwrotnie?
- pochodzenie języka; w jaki sposób zbadać, czy język jest tylko dialektem innego czy też jest zupełnie odrębnym językiem?
- funkcjonalność różnych elementów języka: wiemy np., że instrukcja CALL jest bardziej użyteczna niż GOTO, ale jak zmierzyć istniejącą między nimi różnicę?

Opracowały:
TERESA WÓJCIEKIAN, HALINA CIECHOMSKA
na podstawie referatu Jean E. Sammet,
przedstawionego na konferencji SEAS AM'82

KALENDARZ

Marzec

11—17, Lipsk, ekspozycja sprzętu komputerowego, w ramach Wiosennych Targów Lipskich
12—15, Las Vegas (USA): XII konferencja „Interface '84" na temat transmisji i przetwarzania danych — organizator: The Interface Group, Inc.
19—21, San Diego (USA): międzynarodowe konferencje na temat przetwarzania mowy i sygnałów — organizator IEEE
21—23, Zurych (Szwajcaria): międzynarodowe sympozjum na temat wydajności komputerowych systemów telekomunikacyjnych — organizator: IBM Zürich Research Laboratory
26—28, Karlruhe (RFN): konferencja na temat architektury i eksploatacji systemów komputerowych — organizator: Gesellschaft für Informatik
26—29, Orlando (USA): VII międzynarodowa konferencja na temat inżynierii oprogramowania — organizator: IEEE

Kwiecień

11—13, Paryż: STACS — konferencja na temat teoretycznych aspektów informatyki — organizator: 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

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, Sophia—Antipolis (Francja): konferencja na temat jakościowej oceny predyspozycji w programowaniu — organizator: AFCET
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

30—2.8, Montreal (Kanada): VII międzynarodowa konferencja na temat rozpoznawania obrazów — organizator: International Association for Pattern Recognition

Wrzesień

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

Rada ds. Zastosowań Środków Techniki Obliczeniowej

Współpraca krajów RWPG przy opracowaniu, produkcji i wykorzystaniu środków techniki obliczeniowej odbywa się od 1969 roku pod nadzorem Międzyrządowej Komisji Współpracy w dziedzinie ETO (MK ETO). Działalnością Komisji objęto wszystkie główne fazy realizacyjne ETO, poczynając od prac badawczych, przez konstrukcję, opracowanie technologii wytwarzania, produkcję środków technicznych i wdrażanie systemów — do ich zastosowań i obsługi (wraz z zagadnieniami ekonomicznymi). Organami wykonawczymi MK ETO są Rady: Głównych Konstruktorów JS, Głównych Konstruktorów SM, Ekonomiczna, ds. Zastosowań, ds. Kompleksowej Obsługi, ds. Bazy Elementowej i ds. Normalizacji, a także Tymczasowa Grupa Robocza ds. Urządzeń Technologicznych.

Problematyka oprogramowania użytkowego i zastosowań komputerów jest domeną Rady ds. Zastosowań Środków Techniki Obliczeniowej, utworzonej w 1976 r. Podstawowym zadaniem Rady jest realizacja jednolitej polityki technicznej oraz koordynacja współpracy krajów w dziedzinie zastosowań JS i SM EMC. Koordynacja obejmuje m. in. następujące kierunki:

- określanie najważniejszych dziedzin gospodarki narodowej, gdzie zastosowania środków techniki obliczeniowej przynoszą największe efekty ekonomiczne i społeczne, a także współpraca w przygotowaniu planów perspektywicznych i prognoz zastosowań JS i SM
- opracowanie typowych systemów, rozwiązań projektowych i programów użytkowych, wraz z eksperymentalnym sprawdzeniem ich na wybranych obiektach
- wydawanie uzgodnionych materiałów metodologicznych i zaleceń, dotyczących opracowywanych w ramach MK ETO typowych systemów, rozwiązań projektowych i programów użytkowych
- opracowanie metod i środków automatyzacji projektowania zautomatyzowanych systemów
- opracowanie środków zwiększających efektywność opracowania programów użytkowych oraz automatyzacja prac programistycznych
- opracowanie zasad specjalizacji w dziedzinie systemów zorientowanych obiektowo lub problemowo, z uwzględnieniem zainteresowań, możliwości i doświadczeń poszczególnych krajów
- opracowanie propozycji reguł wymiany pakietów programów użytkowych oraz projektów typowych systemów (w ramach MK ETO).

Główną formą działalności Rady jest realizacja prac merytorycznych, ujętych w tzw. Jednolitym Planie Współpracy (JPW). Dwoma podstawowymi typami prac są: opracowania badawcze oraz pakiety programów użytkowych. JPW, przyjęty w 1981 roku na okres do 1985, zawierał ok. 190 tematów: strona polska prowadziła 23 tematy jako kraj wiodący, a w 63 tematach uczestniczyła jako współwykonawca.

Nadzór nad realizacją prac sprawują organy robocze Rady, tj.:
— Sekcja ds. Systemowych i Metodologicznych
— Sekcja ds. Systemów Automatyzacji Projektowania (SAPR)
— Tymczasowa Grupa Robocza (TGR) ds. Zautomatyzowanych Systemów Sterowania Procesami Technologicznymi (ZSS PT)
— TGR ds. Technologii Projektowania i Narzędzi Programowania (TPNP)
— TGR ds. Przygotowania Kadr (PK).

Tematy dotyczące Zautomatyzowanych Systemów Zarządzania są realizowane bez opieki wyspecjalizowanego organu roboczego.

Podkreślić należy, że — oprócz wspólnej realizacji prac badawczych i rozwojowych — Rada ds. Zastosowań ŚTO prowadzi także prace organizacyjne, które skupiają się ostatnio na tworzeniu w krajach RWPG rynku oprogramowania użytkowego i na wprowadzaniu form współpracy opartych na kontraktach lub umowach handlowych.

Centralnym problemem działalności Rady ds. Zastosowań ŚTO jest rozwój efektywnych zastosowań informatyki, a w szczególności — rozwój projektowania systemów informatycznych i oprogramowania użytkowego JS i SM EMC. Wykorzystanie działalności Rady do rozwijania najbardziej efektywnych — gospodarczo i społecznie — zastosowań informatyki w Polsce, jest głównym celem działania polskiej części Rady. Celami ubocznymi są:
— zmniejszenie kosztów rozwoju zastosowań informatyki w Polsce
— promocja eksportu — zarówno oprogramowania, jak i kompletnych systemów — przez realizację tematów i działania organizacyjne
— gromadzenie i udostępnianie informacji o rozwiązaniach zagranicznych.

Ze względu na umiejscowienie sekretariatu polskiej części Rady ds. Zastosowań, szczególną rolę w koordynowaniu jej bieżącej działalności odgrywa Instytut Organizacji Przemysłu Maszynowego. W osiąganiu postawionych celów wykorzystuje możliwości wynikające z jego zadań i uprawnień związanych z zastosowaniami informatyki w hutnictwie i przemyśle maszynowym. IOPM współpracuje m.in. z ELWRO, IKSAiP, ISS, IMM, MERA SYSTEM i ZW MiM ERA. W pracach Rady uczestniczą też inne jednostki resortu hutnictwa i przemysłu maszynowego, m.in. Instytut Obróbki Skrawaniem i OBR TEKOMA. Wśród jednostek współpracujących spoza resortu HiPM należy wymienić: CPiZI, CEKAR, BISTYP, Uniwersytet Warszawski i Zakład PAN w Bytomiu.

Inny charakter ma działalność TGR ds. Przygotowania Kadr. Jest ona bowiem prowadzona przede wszystkim przez SKI oraz wyższe uczelnie: Politechnikę Wrocławską, Uniwersytet Wrocławski, Politechnikę Poznańską Uniwersytet Toruński i Politechnikę Rzeszowską.

Z konkretnych prac warto wymienić następujące:

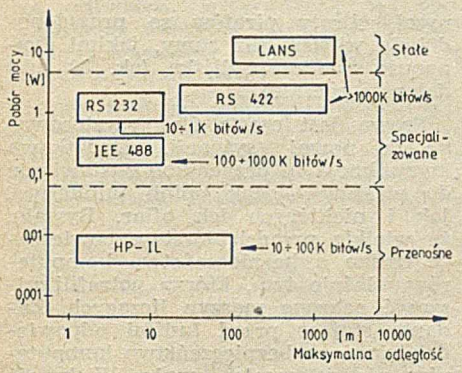
- Instytut Obróbki Skrawaniem tworzy bazę danych technologicznych do optymalizacji parametrów obróbki skrawaniem metali (wraz z oprogramowaniem dla maszyn JS), której wykorzystanie powinno doprowadzić do obniżki kosztów w przedsiębiorstwach należących do kilku gałęzi przemysłu.
- OBR TEKOMA prowadzi prace nad zestawem programów (dla maszyn JS) wykorzystujących metodę elementu skończonego — przyczynią się one do unowocześnienia obliczeń konstrukcyjnych wykonywanych przy projektowaniu maszyn.
- W przedsiębiorstwie MERA-SYSTEM opracowano system komputerowego wspomagania pracy programisty (maszyn JS), służący obniżce kosztów zastosowań informatyki.
- ISS pracuje nad oprogramowaniem systemowym sieci komputerów JS i SM.

KRZYSZTOF URBANIEC

Obsada personalna Polskiej Części Rady ds. Zastosowań Środków Techniki Obliczeniowej:

- **Przedstawiciel PRL w Radzie ds. Zastosowań ŚTO — doc. dr hab. inż. Krzysztof Urbaniec (IOPM ORGMASZ, Warszawa)**
- **Zastępcy Przedstawiciela PRL w Radzie ds. Zastosowań ŚTO — mgr inż. Zbigniew Substyk (CPiZI, Warszawa), mgr inż. Janusz Sieczko (MERA-SYSTEM Warszawa)**
- **P.o. Sekretarza Polskiej Części RT ŚTO — mgr inż. Eugeniusz Kruk (Sekretariat Polskiej Części RZ ŚTO: IOPM Orgmasz; 00-921 Warszawa 53, ul. Krucza 36, tel. 21-36-00, teleks 812720)**
- **Przedstawiciel PRL w Sekcji ds. Ogólnosystemowych i Metodologicznych — doc. dr inż. Jerzy Marszałek (IKSAiP, Wrocław)**
- **Przedstawiciel PRL w Sekcji ds. SAPR — doc. mgr inż. Jerzy Sikora (IOS, Kraków)**
- **Przedstawiciel PRL w TGR ds. ZSSPT — prof. dr hab. inż. Andrzej Grzywak (ISS, Katowice)**
- **Przedstawiciel PRL w TGR ds. TPNP — mgr inż. Marian Skupiński (PSK MERA-SYSTEM, Warszawa)**
- **P.o. Przedstawiciela PRL w TGR ds. PK — mgr Barbara Szymańska (SKI, Warszawa).**

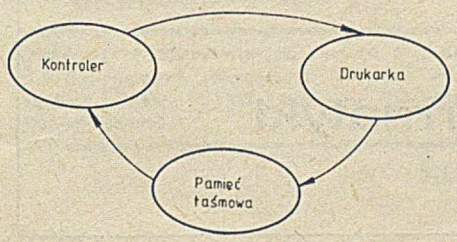
20 grudnia 1981 firma HEWLETT-PACKARD wprowadziła na rynek nowy typ sprzęgu. Został on nazwany HP-IL, gdzie IL oznacza Interface Loop (sprzęg pętlowy). Skąd wziął się pomysł wprowadzenia nowego typu sprzęgu, skoro poprzedni standard, GP-IP (IEEE 488) — też zresztą opracowany przez HEWLETT-PACKARD, jako HP-IB — przyjął się na całym świecie i istnieje już grubo ponad tysiąc typów przyrządów pomiarowych oraz minikomputerów umożliwiających wzajemne połączenie tylko poprzez wetknięcie odpowiedniego wtyku. Otóż sprzęg IEEE 488 jest typu równoległego, co umożliwia jednoczesne kontaktowanie się sterownika ze wszystkimi urządzeniami systemu. Zwiększa to znacznie szybkość przekazywania informacji, ale jednocześnie powoduje też znaczną komplikację systemu kabli łączących (16 przewodów) i duży pobór mocy (p. rys. 1).



Rys. 1. Porównanie HP-IL z innymi systemami sprzęgów (wg Electronic Engineering, October 1982, p. 9)

Nowy typ sprzęgu HP-IL nie ma na celu zastąpienia poprzedniego standardu, a rozszerzenie możliwości łączenia różnych urządzeń pomiarowych i sterowników — rozszerzenie w stronę urządzeń przenośnych oraz podręcznych. By uprościć konstrukcję sprzęgu, zdecydowano się na system pętlowy, w którym każde kolejne urządzenie odbiera i przekazuje dalej wszystkie rozkazy wysłane przez sterownik. Oczywiście, po okrążeniu całej pętli rozkaz powraca do sterownika, który dzięki temu może sprawdzić pracę pętli.

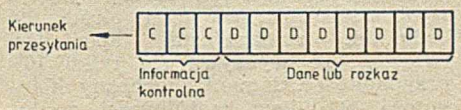
Spróbuję pokrótce opisać zasadnicze elementy tworzące standard nowego sprzęgu oraz sposób przesyłania informacji w pętli. Zainteresowanych samodzielnym dorobieniem sprzęgu typu HP-IL do różnych urządzeń spieszę poinformować, że firma HEWLETT-PACKARD wystąpiła do IEEE o uznanie HP-IL za normę. Tak więc być może już w niedalekiej przyszłości można będzie nabyć odpowiednie układy scalone oraz wtyki tworzące kościec tego sprzętu. Dotychczas HEWLETT-PACKARD udostępnia te wyroby tylko tym wytwórcom aparatury pomiarowej, którzy są w stanie sprostać bardzo wyśrubowanym normom jakościowym. Wszystkim zainteresowanym polecam natomiast książeczkę G. Kane'a, S. Harpera i D. Ushijimy pt. „The HP-IL System" (wyd. Osborne/McGraw-Hill, ISBN 0-931988--77-2). Dużo informacji o HP-IL znaleźć też można w HEWLETT-PACKARD JOURNAL ze stycznia 1983.



Rys. 2. Najprostsza konfiguracja w systemie HP-IL

Najprostsza konfiguracja w systemie HP-IL przedstawiona została na rysunku 2. Sterownikiem może tu być komputer kieszonkowy HP41C lub komputer przenośny HP75.

System HP-IL jest sprzęgiem szeregowym, tzn. informacje są przesyłane kablem dwużyłowym, bit po bicie. Wszystkie urządzenia w pętli komunikują się poprzez rozkazy, które są utworzone przez 11 bitów każda i mają strukturę pokazaną na rysunku 3.

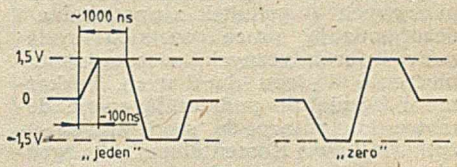


Rys. 3. Struktura komend (11 bitów)

Wyróżniamy trzy typy urządzeń w pętli: sterowniki urządzenia odbiorcze oraz nadawcze. Urządzenia nadawcze są źródłem danych przesyłanych przez pętlę, odbiorcze są urządzeniami otrzymującymi dane z nadawczych oraz rozkazy od sterownika. Rola urządzenia odbiorczego lub nadawczego jest przydzielona każdemu elementowi pętli przez sterownik. Mankamentem standardu HP-IL jest — jak dotychczas — możliwość włączenia w pętli tylko jednego sterownika. Niemożliwe jest więc przyłączenie komputera kieszonkowego HP41 do komputera przenośnego HP75. Sterownik jest odpowiedzialny za przydzielanie wszystkim urządzeniom pętli adresów, odpowiedniej roli oraz inicjowanie przesyłania informacji przez urządzenia informujące do urządzeń odbiorczych.

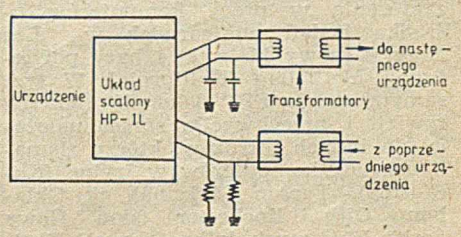
HP-IL ma zadanie łączyć urządzenia przenośne i podręczne, zdecydowano zatem, że pętla może obsługiwać do 31 różnych elementów. W rozszerzonej wersji (adresowanie dwubajtowe) HP-IL może obsługiwać do 960 urządzeń. Szybkość przenoszenia informacji przez pętlę została określona na maksimum 20 KB/s. W dostępnych obecnie urządzeniach szybkość ta jest o rząd mniejsza i wynosi ok. 2 KB/s. Jest to i tak dużo (ok. pół strony maszynopisu na 1 s), i w zupełności wystarcza do obsługi typowych urządzeń zewnętrznych, w jakie wyposaża się komputery osobiste.

Rysunek 4 pokazuje przebiegi sygnałów zera i jedynki; jak widać, zastosowano system trzech poziomów (0 oraz ± 1.5 V). Poziom zerowy jest poziomem braku sygnału, czyli braku aktywności pętli. Same kable sprzęgu są oddzielone transformatorami od urządzeń aktywnych (rys. 5). Transformatory te służą jako urządzenie dopasowujące różne poziomy logiczne różnych urządzeń pętli. Jednocześnie oddzielają one samą linię transmisyjną sygnału od tych różnych poziomów logicznych i ewentualnych szumów generowanych w urządzeniach (silniki, zasilacze etc.).



Rys. 4. Przebiegi sygnałów zera i jedynki

Kable łączące urządzenia w pętli są produkowane w długościach 1—100 m. W przypadku długości 100 m konieczny jest specjalny kabel ekranowany, gdyż zakłócenia zewnętrzne mogłyby uniemożliwiać prawidłową pracę sprzęgu. Każdy kabel zakończony jest z jednej strony wtykiem żeńskim, a z drugiej męskim — dzięki temu niemożliwe jest błędne połączenie pętli (np. wyjście jednego urządzenia z wyjściem następnego).



Rys. 5. Transformatory oddzielające sprzęg od urządzeń aktywnych

Ponieważ HP-IL może być używany do sprzęgania urządzeń zasilanych z baterii, pomyślano o możliwości zdalnego włączania oraz wyłączania poprzez odpowiednie rozkazy przesyłane sprzęgiem. Przewidziano też tryb „stand-by" czyli oczekiwania, gdy urządzenie pobiera moc tylko do podtrzymania pamięci. W ten sposób użytkownik może łatwo skompletować w terenie sieć pomiarową, która będzie automatycznie włączała się o określonej godzinie, dokonywała pomiarów, a potem przechodziła w stan oczekiwania na kolejne dane. Sieci takie już wykorzystuje się w automatycznych stacjach meteorologicznych.

Na zakończenie podajmy przykładowe dane o już produkowanych urządzeniach pomiarowych i peryferyjnych wyposażonych w sprzęg HP-IL. Oprócz wspomnianych już sterowników (HP75 wyposażony jest fabrycznie w odpowiednie gniazda i całą elektronikę sprzęgu) firma HEWLETT-PACKARD produkuje dwa typy drukarek (termiczną oraz iglicową), ploter, pamięć taśmową na mikrokasetach (pojemność 131 KB), samokalibrujący się multimetr cyfrowy oraz przełącznik 30-kanałowy, dzięki któremu można zbierać dane z 30 urządzeń analogowych. Ponadto są dostępne konwertery pomiędzy HP-IL a HP-IB.

Kilka firm produkuje już własne wyroby i wyposaża je w sprzęg HP-IL. Firma OCEAN SCIENTIFIC — na przykład — wytwarza konwerter analogowo-cyfrowy o dokładności 3 1/2 cyfry, wykonujący do pięciu konwersji na sekundę. W ten sposób można bardzo tanio złożyć najprostszy system pomiarowy: sterownik HP41CX (ok. 300 dol.), moduł HP-IL (ok. 100 dol.), konwerter A/D (ok. 400 dol.), pamięć taśmowa (ok. 350 dol.) i drukarka termiczna (ok. 350 dol.).

System taki umożliwia automatyczne wykonywanie pomiarów, zapamiętywanie ich, przetwarzanie, drukowanie wyników, a nawet rysowanie wykresów. Należy dodać, że wszystkie wymienione urządzenia są zasilane z baterii, a cały zestaw łatwo mieści się w teczce. Jeszcze niedawno zestaw o takich możliwościach kosztował ponad 10 tys. dol. i z trudem mieścił się na dużym stole. Cóż to jednak znaczy postęp.

JAKUB TATARKIEWICZ

Piraci informatyki

Prasa zachodnia co jakiś czas przynosi nowe wieści z frontu walki z tymi, którzy z różnych — nie tylko finansowych — pobudek z upodobaniem dezorganizują pracę wielkich systemów informatycznych, niszczą zabezpieczenia dostępu do danych, bezkarnie i bezpłatnie podłączają swoje domowe komputery do sieci innych użytkowników. Komentatorzy są zgodni w ocenie rozmiarów zjawiska, różnice dotyczą interpretacji. Czy mamy do czynienia wyłącznie z kolejnym problemem gospodarczym, który należy rozwiązywać znanymi, skutecznymi sposobami, czy zarysowuje się nowy problem społeczny, wymagający całkiem odmiennego podejścia? Jedno nie ulega wątpliwości (chociaż zainteresowanym niełatwo to przyznać) — zwiększa się bezradność wobec tego zjawiska.

Agenci FBI zlikwidowali w październiku 1983 w Detroit (Michigan) gang młodocianych przestępców — prawdziwych geniuszy informatyki, z których najstarszy miał zaledwie 17 lat. Podejrzewano ich, że za pomocą urządzeń elektronicznych spowodowali zakłócenia w pracy komputerów ośrodka badań nuklearnych w Los Alamos (nowy Meksyk) i bazy powietrznej McClellan (Kalifornia). W tym samym czasie FBI dokonała licznych rewizji i przechwyciła sprzęt należący do innych intruzów, którzy złamali ochronę komputerów w Irvine (Kalifornia), Tucson (Arizona), w Oklahoma City i Rochester (stan Nowy Jork). W samych tylko Stanach Zjednoczonych odnotowano liczne przypadki zakłócania pracy komputerów. Wielkie przedsiębiorstwa informatyczne mają nawet specjalną kartotekę osobników podejrzanych o taką działalność.

W Zachodniej Europie także mówi się o piratach informatyki, m.in. przy okazji „przecieków", w wyniku których ujawniono wewnętrzną organizację Unii Banków Szwajcarskich oraz stan kont pewnych klientów zagranicznych, przeważnie francuskich. Pozwoliło to francuskim służbom celnym ułożyć listę oszustów podatkowych nielegalnie wywożących kapitały za granicę.

Przywódca amerykańskiego gangu nie był wcale szpiegiem, bo pozostawiał w komputerze informacje o sobie i swojej działalności. Można jednak wyobrazić sobie, że nowi piraci nie będą tak romantyczni. Specjaliści twierdzą, że wykrywa się tylko 10% przypadków „złamania" ochrony komputera. Nie ulega wątpliwości, że ludzi zabawiających się zakłóceniem pracy komputerów jest znacznie więcej niż zawodowych szpiegów. Uważa się, że przypadki świadomego oszustwa stanowią 43% „fałszerstw informatycznych", 23% dotyczy kradzieży kart perforowanych zawierających informacje, a w 15% piraci przeprowadzają potrzebne im operacje obliczeniowe.

Ludzie odpowiedzialni za zapewnienie bezpieczeństwa komputerów oceniają na 15 mld franków roczne straty wynikające z uszkodzenia lub bezprawnego wykorzystywania maszyn liczących — i to tylko na terenie Europy. W Stanach Zjednoczonych, gdzie banki przekazują codziennie ok. 400 mld dolarów metodami informatycznymi, obroty piratów są prawdopodobnie większe niż sumy, jakimi dysponuje mafia sycylijska (ok. 30 mld dolarów rocznie).

Prawo jest często źle przystosowane do nowej sytuacji. Dodatkowa komplikacja to przychylne nastawienie do piratów — tak opinii publicznej, jak i niektórych ich ofiar. Bywało, że wielkie przedsiębiorstwa powierzały troskę o bezpieczeństwo komputerów właśnie tym, którzy potrafili zniszczyć ochronę maszyn liczących. Zadania stojące przed ludźmi odpowiedzialnymi za bezpieczeństwo komputerów nie są wcale małe. Większość wielkich komputerów jest przyłączona do sieci, a więc całego systemu obiegu informacji, co ogromnie zwiększa ryzyko szkód. Złamanie kodu zabezpieczającego komputer to — według samych piratów — często kwestia cierpliwości i odrobiny szczęścia. Tym bardziej że — jak potem wykazuje dochodzenie — ludzie odpowiedzialni za bezpieczeństwo maszyn liczących sami pozostawiają nieraz klucze kodowe albo zapominają zmienić kody po upływie dłuższego czasu. Poza komputerami używanymi przez resort obrony, zabezpieczenie maszyn liczących nie jest zbyt skomplikowane. Często stosowane są tylko karty z zapisem magnetycznym, których posiadanie umożliwia pełne lub częściowe wykorzystywanie komputera.

Wielcy producenci komputerów poświęcają już 20% sum przeznaczonych na badania i rozwój właśnie na rozwiązanie kwestii bezpieczeństwa. Rozważa się skuteczność np. dodatkowego kodowania zleceń przekazywanych komputerowi lub fizycznej identyfikacji operatora na podstawie odcisków palców albo tonu jego głosu.

Prof. Don Parker z uniwersytetu Stanford (Kalifornia) stwierdził autorytatywnie, że w najbliższym dziesięcioleciu należy obawiać się nie tyle wojny atomowej, co wojny elektronicznej... „Jakiś kraj może podporządkować sobie drugi, jeśli uda mu się zablokować komputery przeciwnika".

K.I.

IFIP

Międzynarodowa Federacja Przetwarzania Informacji (International Federation for Information Processing — IFIP) została utworzona w Paryżu w roku 1960 w wyniku inicjatywy delegacji ośmiu krajowych stowarzyszeń informatycznych, które współpracowały z UNESCO przygotowując — odbyty tam rok wcześniej — pierwszy Światowy Kongres Informatyki. Obecnie IFIP jest międzynarodową federacją zawodowych i technicznych organizacji (lub krajowych związków takich organizacji) zajmujących się problemami przetwarzania informacji. Z każdego kraju tylko jedna organizacja — reprezentatywna dla jego działalności w tej dziedzinie może stać się członkiem rzeczywistym. W dniu 1 stycznia 1983 roku członkami federacji były 43 organizacje krajowe (z Polski — Polska Akademia Nauk).

Celem IFIP jest wspieranie rozwoju nauki i techniki przetwarzania informacji oraz rozszerzanie współpracy międzynarodowej w tej dziedzinie. Federacja wspomaga i skupia wszelkie działania, które pod względem naukowym i technicznym podnoszą poziom przetwarzania informacji. IFIP prowadzi swoje prace w ramach 10 Komitetów Technicznych i 33 Grup Roboczych, które obejmują swym działaniem główne działy informatyki. Jednostki te organizują konferencje, sympozja, seminaria oraz formalne i nieformalne spotkania.

Najwyższą władzą federacji jest Zgromadzenie Ogólne, które zbiera się raz do roku. Każda organizacja członkowska ma w nim jednego reprezentanta. Zgromadzenie decyduje we wszystkich istotnych sprawach, takich jak ogólna polityka, program przedsięwzięć, przyjęcia, wybory i budżet. Codzienną pracą IFIP kieruje Zarząd składający się z przewodniczącego, trzech zastępców, sekretarza i skarbnika, wybranych przez Zgromadzenie Ogólne. Dwa razy w roku spotyka się Rada, podejmująca decyzje niezbędne w okresie pomiędzy posiedzeniami Zgromadzenia. W skład Rady wchodzą członkowie Zarządu i wybierani ze składu Zgromadzenia jej członkowie, których liczba nie przekracza 8. Siedzibą federacji jest Genewa (Szwajcaria) i tam też znajduje się jej sekretariat, administrujący bieżącą działalnością.

Adres: IFIP Secretariat, 3, rue du Marché, CH-1204 Généve, Szwajcaria. Telefon: (022) 28-26-49, teleks: 428 472 ifip ch, telegraf: ifipsec geneva.

Oprac. MkS

Paradoks ergonomii

SICOB 1983, czyli Międzynarodowy Salon Informatyki, Telekomunikacji, Organizacji i Automatyzacji Prac Biurowych (Paryż, wrzesień 1983) zajmował się — jak nigdy dotąd — problemami ergonomii.

Dużo wysiłku włożono w obniżenie hałaśliwości drukarek. Większość tych urządzeń wyposażono w ruchome, superpłaskie klawiatury. Przyciski są zgrupowane według funkcji, które spełniają — tak, aby palec trafił na nie jak najszybciej. Ekrany pokryto emulsjami przeciwodblaskowymi i można je ustawiać pod dowolnym kątem w każdej płaszczyźnie. Szwedzka firma FACIT proponuje nawet terminal TWIST, którego ekran może być wykorzystany tradycyjnie (dłuższy bok poziomio) albo po obróceniu o 90° (pionowo) i wtedy mieści się na nim odpowiednik kartki papieru formatu A4. Rozmaite dodatkowe funkcje usprawniają odczytywanie i wymianę informacji. Migotanie, podkreślanie, obraz negatywowy, zmienne kroje znaków, a wreszcie — wielobarwność czy grafika, wszystko to pozwala zwracać uwagę na najważniejsze fragmenty obrazu (błędy, tytuły), odróżniać grupy danych (stałe — zmienne, wejściowe — wyjściowe). Przykładem urządzenia o takich możliwościach jest ALFASKOP 4113 firmy ERICSSON, za pomocą którego można konstruować oprogramowanie również zorientowane na ergonomię. Oczywiście, monitory tego typu są wspierane przez drukarki, odtwarzające na papierze wszystkie subtelności uzyskanego obrazu jak kolor, krój znaków itd.

Wszystkie te udogodnienia (wielokrotne okienka na ekranie, „myszki" itp.) prezentowano zresztą już wcześniej, przed Salonem. W walce o klienta wygrywają ci, którzy zaproponują najlepszą dokumentację, najprostszy sposób użytkowania, najodpowiedniejszą i najszerszą gamę programów.

Można było zatem mieć nadzieję na znalezienie także wielu systemów wspomagających programowanie — zmniejszających wysiłek programistów. Trzeba się było jednak dobrze naszukać.

W cieniu głównego stoiska firmy APPLE prezentowano system programowania w pełni wykorzystujący możliwości graficzne i „myszkę" LISY. Firma SYSECA zaproponowała narzędzie do programowania w PASCALU oraz LAQUAD — przeznaczony dla większych komputerów system dokumentacji i kontroli jakości SYMPA. Z kolei „warsztat oprogramowania" MULTIPRO (grupa CAP SOGETI) oferował całą gamę takich systemów o

W SKRÓCIE ● W SKRÓCIE

■ Zachodnioniemieckie stowarzyszenia informatyczne — GI (Gesellschaft für Informatik) oraz GMD (Gesellschaft für Mathematik und Datenverarbeitung) — są inicjatorami i organizatorami młodzieżowych konkursów programowania. Ich celem jest zachęcanie młodzieży do sformalizowanego rozwiązywania problemów z uwzględnieniem realiów zastosowań oraz opanowania umiejętności przejrzytego dokumentowania programów. Kolejny konkurs z tej serii trwał od kwietnia 1982 do lutego 1983 a jego plonem było 187 prac. Warunki uczestnictwa to — odpowiedni wiek (osoby urodzone po 31.12.1963), niestudiowanie na kierunkach informatycznych oraz brak formalnego przeszkolenia w zawodach informatycznych. Wybór języka programowania i sprzętu komputerowego był całkowicie dowolny. Nadesłane prace dotyczyły głównie gier komputerowych, matematyki, zarządzania i grafiki komputerowej, a najczęściej użytymi językami były BASIC i PASCAL. (K)

*

■ Jeszcze w czerwcu ub. r. pisano o tym jako o projekcie na przyszłość. Tymczasem w końcu października rozpoczęła się akcja marketingowa tanich komputerów domowych (200—300 dol.) przygotowanych przez siedem japońskich koncernów — MATSUSHITA, TOSHIBA, SONY, SANYO, MITSUBISHI, NIPPON GALEKI i HITACHI. Wszystkie wyposażono w co najmniej 16 KB pamięci RAM oraz rozszerzony BASIC firmy MICROSOFT, zajmujący 32 KB pamięci ROM. Jest to oczywiście realizacja standardu sprzętowo-programowego MSX stworzonego przez MICROSOFT we współpracy z producentami japońskimi. Przewiduje się, że głównymi nabywcami będą uczniowie szkół podstawowych i średnich. Po pierwszej serii (100 tys. sztuk w końcu ubiegłego roku), produkcja ta wyniesie 500 tys. szt. i milion szt. w latach 1984 i 1985. Również wytwórcy oprogramowania spodziewają się szybkiego rozwoju rynku i planują wzrost liczby programów zgodnych ze standardem MSX z 30 — pod koniec ub.r. do 500 — na początku tego roku. (M)

*

■ Być może z większym powodzeniem niż na rynku TEXAS INSTRUMENTS będzie walczyć przed sądem. Na przełomie lat sześćdziesiątych i siedemdziesiątych, gdy TI prowadziła pionierskie prace w dziedzinie jednoukładowych kalkulatorów, opatentowano kilka rozwiązań. Dwa patenty dotyczyły techniki związanej z komunikacją z urządzeniem peryferyjnym (tzw. bit-pusher-technique). Trzeci — obejmował urządzenie będące kombinacją ręcznego urządzenia wejściowego (jakim jest np. klawiatura), mikrokomputera jednoukładowego oraz wyświetlacza. Według analizy prawników z TI, patenty te dotyczą poważnej części komputerów osobistych. Obecnie przygotowuje się kroki, które mają skłonić innych producentów do honorowania tych patentów. Pierwszy atak powiódł się. Założona w 1982 roku przez byłych pracowników TI firma COMPAQ zgodziła się uiszczać opłaty licencyjne. (M)

W SKRÓCIE ● W SKRÓCIE

W SKRÓCIE ● W SKRÓCIE

■ Następuje dalsza obniża cen sprzętu mikrokomputerowego. W Anglii SPECTRUM (48 K) kosztowała we wrześniu ub.r. 130 funtów a 1 EK — 100 funtów. W Stanach Zjednoczonych wszystkie mikrokomputery SINCLAIR są wyposażone w typowe klawiatury i nazywają się TIMEX SINCLAIR. ZX81 z pamięcią 16 K był sprzedawany za 90 dol, a odpowiednik SPECTRUM z pamięcią 40 K za 150 dol. i 72 K za 200 dol. Wersja podstawowa ZX81 kosztuje w Stanach Zjednoczonych 30 dol. Również firmy amerykańskie obniżają swe ceny: APPLE III kosztuje 2695 dol., IBM PC — w wersji bazowej — 1864 dol., a COMMODORE 64 — mniej niż 300 dol. (A)

*

■ Grupa pracowników IEEE zwróciła się w 1983 roku do rządów Stanów Zjednoczonych o dotację na budowę superkomputera o prędkości 200-krotnie przewyższającej możliwość komputerów CYBER-205 oraz CRAY X-MP. Budowa tego superkomputera, o szybkości 80 mld operacji/s, ma trwać pięć lat, a koszty całego przedsięwzięcia oceniono na 500 mln dol. Równolegle z pracami konstrukcyjnymi prowadzone będą prace nad oprogramowaniem wspomagającym działanie komputerów wieloprocesorowych. (W)

*

■ Prawdziwie sensacyjna wiadomość pojawiła się w ostatnich dniach listopada 1983. Ogłosiła ją stosunkowo nowa, bo o zaledwie rocznej działalności, firma komputerowa ETA SYSTEMS INC. Firma ta, składająca się z dziesięcioosobowej grupy byłych pracowników CDC, ma zamiar kontynuować dobre tradycje komputera CYBER-205 i stworzyć, jako jego następcę, maszynę o prędkości 10 mld operacji zmiennoprzecinkowych na sekundę. Nie wiadomo jakie są koszty, nie podano też terminu zakończenia prac. Wiadomo jednak, że konstruktorzy będą posługiwali się superkomputerem CYBER-205, a w pracach projektowych będzie uczestniczyć ponad sto osób. (W)

*

■ Komputery mogą służyć kompozytorom utworów muzycznych. Ostatnio na rynku zachodnim pojawił się pakiet programowy MUSIC CONSTRUCTION SET, reklamowany jako urządzenie ułatwiające komponowanie, umożliwiające naukę sztuki kompozycji oraz dające możliwość wykonywania utworów muzycznych. Odpowiednie oprogramowanie pozwala sterować wyświetlanym na ekranie obrazem ręki. Manetka (ang. joystick), klawiatura oraz specjalne przyciski umożliwiają wykonywanie utworów muzycznych oraz zapisywania melodii poprzez wprowadzanie wpisanych w obraz pięciolinii symboli odpowiednich znaków muzycznych. Kompozytor może nakładać na siebie różne wątki melodyczne i rytmy — regulując ich natężenie, tempo oraz barwę tonu. Melodie mogą być zapamiętane na dysku. Oprogramowanie zostało opracowane dla komputerów APPLE II, II+, IIe. (W)

W SKRÓCIE ● W SKRÓCIE

szerokich możliwościach: zarządzanie bibliotekami zastosowań wraz z historią modyfikacji i kontroli, generator systemów i formularzy ułatwiający tworzenie dokumentacji, generator programów (kod i instrukcje sterujące) i śledzenie prac w toku (stan zadań, planowanie, metoda PERT).

Systemy te mogą w istotny sposób ułatwić prace nad zastosowaniami informatyki, oczywiście pod warunkiem, że same będą łatwe do opanowania i wykorzystania. Projektanci włożyli sporo wysiłku w ergonomię sprzętu i można mieć nadzieję na rozpowszechnienie się takiej postawy.

Może jednak następny — SICOB 1984 — będzie okazją do jeszcze wyraźniejszego uświadomienia wszystkim faktu, że ergonomia jest potrzebna nie tylko użytkownikom, ale również zawodowym informatykom tworzącym ergonomiczne oprogramowanie.

Oprac. MkS
na podstawie 01 HEBDO z 10.10.1983

Wizja bio-kostki

Futurolodzy, którzy kiedykolwiek borykali się z problemem stworzenia komputera opartego na strukturach białkowych zamiast tradycyjnych kostek krzemowych, długo czekali na wydarzenie tej miary, co niezwykłe spotkanie naukowców w Los Angeles w drugiej połowie października 1983. Niektórzy badacze twierdzą, bowiem, że tak zwany molekularny komputer, który przejąłby funkcje ludzkiego mózgu, nie jest wcale mrzonką. Jednakże wyniki konferencji powinny ostudzić nieco zapał entuzjastów, przekonanych, że zastosowanie „bio-kostki" jest kwestią niedalekiej przyszłości.

Trzydziestu pięciu uczestników spotkania miało szerokie pole do popisu. Po pierwsze — reprezentowali co najmniej cztery dziedziny wiedzy: informatykę, chemię, fizykę i biologię; po drugie — w swych rozważaniach nie brali pod uwagę finansowych aspektów ewentualnego wdrożenia nowych technologii. Zadanie było jasno określone: ocenić możliwości skonstruowania molekularnego komputera i przedstawić Narodowej Fundacji Naukowej (NSF) raport zawierający odpowiedź na pytanie, czy warto inwestować w realizację pojawiających się coraz częściej pomysłów stworzenia bio-kostki. Chociaż raport miał być gotowy pod koniec roku 1983, już w chwili zakończenia obrad było jasne, że molekularny komputer wymaga jeszcze badań podstawowych, a mówienie o jego zastosowaniach jest zdecydowanie przedwczesne. Pojawiły się jednak głosy, że ton sprawozdania z konferencji nie powinien być zbyt pesymistyczny — tak, by nie zniechęcać do dalszych poszukiwań.

Wymogi praktyczne stwarzają konieczność prowadzenia wielu badań, a nie tylko rozważań teoretycznych — teoretycznie bowiem komputer molekularny powinien działać. Stworzenie trójwymiarowej konfiguracji struktur białkowych, np. enzymów, to możliwość skonstruowania maszyny klasyfikującej, gromadzącej i przetwarzającej informacje w sposób bardziej złożony niż dwustanowe przełączniki maszyn cyfrowych. Miałaby ona własny rodzaj inteligencji, zdolność kojarzenia, rozpoznawania obrazów i obiektów, podejmowania decyzji.

Przeszkody, które trzeba pokonać są ogromne. Jak połączyć molekularne przełączniki w funkcjonującą sieć? Jak stabilizować cząsteczki mające tendencję do samozniszczenia (należałoby zwiększać ich rozmiary, co dałoby jednak przewagę tradycyjnym strukturom krzemowym). Jak imitować systemy biologiczne (nie są dokładnie znane właściwości organicznych półprzewodników, nie rozwiązano problemu eliminowania zanieczyszczeń)?

Uczestnicy konferencji zdecydowanie natomiast poparli rozwój prac nad bio-sensorami, urządzeniami tworzonymi na bazie struktur białkowych i mogącymi klasyfikować dane według określonych schematów. Można je przyłączyć do komputerów, które dokonują zasadniczej analizy.

K.I.

Nadesłane sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego

Nr 121. Bolesław Kacewicz: Optymality of Euler — integral information for solving a scalar autonomous ODE

Nr 122. Maksymilian Dryja: The k-Reducion-Matrix Decomposition Algorithm for Diference Eliptic Problem. A Decomposition Method for Solving Finite Element Equations

Nr 123. Marek A. Kowalski, Zbigniew Sawoń: The problem of moments in normed spaces

Nr 124. Alicja Smoktunowicz, Jolanta Sokolnicka: Realizacja algorytmu BCIR w arytmetykach podwajanych mantys

Nr 125. Joanna Wiśniewska: Wybrane informacje o systemie operacyjnym RSX-11M

Nr 126. Lech Banachowski, Grażyna Drozdowicz: On the average complexity of binary search trees

Nr 128. Wojciech Rytter: Złożoność czasowa dwukierunkowych automatów stosowych i programów rekurencyjnych

Dokumentacja

Wydaje się, że warto poświęcić nieco uwagi tak ważnej, choć rzadko docenianej sprawie, jaką jest dokumentacja systemu komputerowego. W normie ISO 2382/V (Data Processing Vacabulary) przez **dokument** rozumie się nośnik danych i dane zarejestrowane na nim, na ogół na stałe, które mogą być odczytywane przez człowieka lub komputer. Według normy PN-71/T-01016, **nośnik danych** jest to materiał, w którym lub na którym określona zmienna fizyczna może reprezentować dane.

Zwróćmy uwagę, że w definicji dokumentu nie powiedziano wyraźnie, być może świadomie, że zarejestrowane dane muszą być czytelne dla człowieka. Moim zdaniem, jest to nieodłączna cecha dokumentu, bo na cóż mi „dokument" w postaci kodu wynikowego, którego w żaden sposób nie potrafię odczytać. Prawdopodobnie dlatego, do definicji dokumentu w normie IEEE Std. 729 dodano wyjaśnienie, że termin ten jest często używany tylko w znaczeniu „czytelny dla człowieka".

Dokumentacja (ang. documentation) jest to zbiór dokumentów na określony temat (a collection of documents on a given subject). Natomiast, przez **dokumentowanie** (ang. documentation) rozumie się operowanie dokumentami, które może polegać na ich identyfikowaniu, zbieraniu, przetwarzaniu, przechowywaniu i rozpowszechnianiu (ISO 2382/V).

Dość ważnym pojęciem z tej dziedziny jest tzw. **poziom dokumentacji** (ang. level of documentation). Według normy IEEE Std. 729, jest to opis wymaganej dokumentacji określający jej zakres, zawartość, format i jakość. Wyboru poziomu można dokonać w oparciu o koszty projektu, przewidywane zastosowanie, wielkość nakładów lub inne czynniki.

O ile znaczenia pojęć **dokument** lub **dokumentowanie** są jasne nawet bez podawania definicji, to o poziomach dokumentacji należy powiedzieć coś więcej. W raporcie „Management Guide for Software Documentation" (NBS Special Publication 500-87, National Bureau of Standards, Washington, 1982) określono cztery poziomy dokumentacji programowej, zróżnicowane pod względem szczegółowości.

Poziom 1 (minimalny) jest odpowiedni dla pojedynczych programów powstałych w okresie krótszym od miesiąca. Dokumentacja zawiera **wydruk programu, notatki projektowe** (ang. development notes), **danc testowe i abstrakt programu.** Przez **abstrakt programu** rozumie się (IEEE Std. 729) krótki opis programu komputerowego, zawierający wystarczającą informację dla potencjalnych użytkowników, w celu określenia odpowiedniości programu do ich potrzeb i zasobów.

Poziom 2 (wewnętrzny) odnosi się do specjalizowanych programów, które — według przeprowadzonego dokładnego rozeznania — nie mają szerszego zastosowania. Ich dokumentacja zawiera więcej komentarzy w wydruku programu niż poziom 1, co ułatwia modyfikację oraz użytkowanie programu. Dokumentacja opisowa jest bardzo skąpa.

Poziom 3 (roboczy) stosuje się do programów używanych przez różne osoby w tej samej instytucji a nawet w innych instytucjach. Dokumentacja ma charakter roboczy i jest sporządzona i rozpowszechniona w postaci maszynopisów z minimalnym opracowaniem edytorskim (tzw. working paper).

Poziom 4 (publikacyjny) odnosi się do programów przeznaczonych do powszechnego użycia oraz programów stanowiących przedmiot publikacji naukowych. Strona formalna dokumentacji musi być zgodna z wymaganiami i normami opracowującej instytucji.

Istnieje wiele rodzajów dokumentacji. Najważniejsze z nich są zdefiniowane w normie IEEE Std. 729. **Dokumentacja systemu** (ang. system documentation) opisuje wymagania, filozofię rozwiązania projektowego, szczegóły projektowe, możliwości funkcjonalne, ograniczenia i inne właściwości systemu informatycznego. **Dokumentacja oprogramowania** (ang. software documentation) zawiera dane lub informacje techniczne, włącznie z wydrukami komputerowymi, w postaci czytelnej dla człowieka, które specyfikują projekt i jego szczegóły, wyjaśniają możliwości funkcjonalne oraz stanowią instrukcję eksploatacji służącą do użytkowania oprogramowania zgodnie z jego przeznaczeniem. **Dokumentacja użytkowa** (ang. user documentation) zawiera instrukcje umożliwiające eksploatację systemu informatycznego przez użytkowników zgodnie z jego przeznaczeniem. Przykładem dokumentacji użytkowej jest podręcznik użytkowania (ang. user's manual).

W Stanach Zjednoczonych powstały nawet specjalne normy zawierające wskazówki dotyczące zawartości różnych rodzajów dokumentacji. Norma FIPS-64 (FIPS — Federal Information Processing Standard) określa zalecenia dotyczące trzech rodzajów dokumentów tzw. fazy inicjowania projektu (Project Request, Feasibility Study, Cost Benefit Study), a norma FIPS-38 zawiera wymagania dotyczące postaci wszystkich dokumentów dotyczących cyklu rozwojowego programowania — od sformułowania wymagań do testowania. Liczba wszystkich dokumentów ujętych w wymienionej normie jest znaczna, a przykładowe, dokumenty fazy końcowej, tj. testowania, zdefiniowano następująco:

— plan testowania (ang. test plan) — dokument opisujący podejście do przewidywanych działań testujących (przedmiot i zakres testowania, harmonogram, wymagania osobowe i sprawozdawcze, kryteria oceny itp.)

— sprawozdanie z testów (ang. test report) — opisujący przebieg i wyniki testowania systemu informatycznego lub jego składników.

JANUSZ ZALEWSKI

KONFERENCJE

Minikomputery w automatyce i technice systemów

W dniach 18—21 września br. odbędzie się w Politechnice Wrocławskiej ogólnopolska konferencja naukowo-techniczna MIKROKOMPUTERY W AUTOMATYCE I TECHNICE SYSTEMÓW. Tematyka Konferencji obejmuje zagadnienia architektury i projektowania systemów mikrokomputerowych, oprogramowania podstawowego i użytkowego mikrokomputerów, sieci mikrokomputerowych oraz zastosowań mikrokomputerów w automatyce i technice systemów (w tym do modelowania, symulacji i identyfikacji systemów), w systemach automatyki przemysłowej, systemach pomiarowo-kontrolnych, systemach telekomunikacji i systemach przetwarzania danych, do projektowania systemów automatyki, w automatyzacji badań eksperymentalnych, w systemach biomedycznych do przetwarzania danych i diagnostyki oraz do nauczania automatyki i techniki systemów (laboratoria dydaktyczne).

Konferencję organizuje Instytut Sterowania i Techniki Systemów Politechniki Wrocławskiej, przy współudziale: Komitetu Automatyki PAN (sekcja sterowania procesów dyskretnych), Polskiego Komitetu ds. Pomiarów i Automatyki NOT (sekcja szkolenia Podkomitetu Automatyki), Instytutu Komputerowych Systemów Automatyki i Pomiarów we Wrocławiu, Zakładów Elektronicznych ELWRO we Wrocławiu oraz przedsiębiorstw AMEPROD w Poznaniu i NOWATECH w Katowicach.

Przewiduje się zorganizowanie wystawy sprzętu mikrokomputerowego.

Termin zgłoszenia referatów wraz ze streszczeniami upłynął z końcem lutego br., jednakże organizatorzy rozważą propozycje dodatkowych referatów, o ile będą one zgłoszone nie później niż do końca maja br.

Bliższych informacji udziela dr inż. Leszek Borzemski; Instytut Sterowania i Techniki Systemów Politechniki Wrocławskiej; Wrocław, ul. Wybrzeże Wyspiańskiego 27; tel. 22-06-84, 20-29-92.

Sobczyk M., Szałas A.: PROLOG (1)

INFORMATYKA 1984, nr 3, s. 1

Pierwsza część charakterystyki języka PROLOG, zawierająca zwięzłą historię jego powstania i rozwoju oraz omówienie ulepszonej wersji o nazwie PROLOG II, ilustrowane poglądowym przykładem sposobu programowania i konstrukcji programów.

Lewoc J. B.: Quo vadis, Jednolity Systemie?

INFORMATYKA 1984, nr 3, s. 5

Krytyczne uwagi na temat efektywności firmowego oprogramowania komputera R-32 oparte na przykładzie wyników badania istniejących rozwiązań systemów sterujących teleprzetwarzania i wielodostępu. W zakończeniu autor podaje wnioski i propozycje zmierzające do poprawy sytuacji.

Mirkowski J., Piątkowski A.: Laboratorium dydaktyczne w systemie CAMAC z rozłożoną inteligencją

INFORMATYKA 1984, nr 3, s. 11

Charakterystyka rozwiązania laboratorium opartego na systemie CAMAC do prowadzenia eksperymentów naukowych i dydaktycznych oraz automatyzacji pracowni studenckich. Omówiono konfigurację sprzętową i oprogramowanie oraz przykładową realizację laboratorium, wprowadzonego do procesu dydaktycznego na Politechnice Warszawskiej.

Gacek R., Gościński A., Rempała Z., Rygał R.: Analizator ATA-77 w sytemie CAMAC do analizy dwuparametrycznej

INFORMATYKA 1984, nr 3, s. 21

Charakterystyka analizatora ATA-77 przeznaczonego do automatyzacji eksperymentów w laboratorium wysokich napięć. Omówiono konfigurację, sposób działania i oprogramowanie systemu, a także jego zalety i wady.

Собчик М., Шалас А.: PROLOG (1)

INFORMATYKA 1984, № 3, стр. 1

Первая часть характеристики языка PROLOG, содержащая краткую историю его возникновения и развития, а также обсуждение усовершенствованной версии — PROLOG II, иллюстрированное наглядным примером способа программирования и конструкции программ.

Левоц Й. Б.: Quo vadis, Единая Система?

INFORMATYKA 1984, № 3, стр. 5

Критические замечания на тему эффективности фирменного программного обеспечения ЭВМ R-32, основаны на примере результатов исследования существующих решений управляющих систем телеобработки и многодоступа. В заключение автор делает выводы и предложения направленные к улучшению положения.

Мирковски Й., Пионтковски А.: Дидактическая лаборатория в системе CAMAC с дистрибутивным интеллектом

INFORMATYKA 1984, № 3, стр. 11

Характеристика решения лаборатории, основанной на системе CAMAC, для ведения научных и дидактических экспериментов и автоматизации студенческих лабораторий. Обсуждены аппаратная конфигурация и программное обеспечение, а также пример реализации лаборатории, введенной в дидактический процесс в Варшавском Политехническом Институте.

Гацек Р., Госьциньски А., Рempала З., Рыгал Р.: Анализатор АТА-77 в системе CAMAC для двухпараметрического анализа

INFORMATYKA 1984, № 3, стр. 21

Характеристика анализатора АТА-77 предназначенного для автоматизации экспериментов в лаборатории высоких напряжений. Обсуждаются конфигурация, способ действия и программное обеспечение системы, а также ее преимущества и недостатки.

Sobczyk M., Szałas A.: PROLOG (1)

INFORMATYKA 1984, Nr. 3, S. 1

Erster Teil einer Charakteristik von PROLOG Programmiersprache, der eine kurzgefasste Geschichte ihrer Entstehung und Entwicklung, sowie eine Beschreibung der verbesserten, PROLOG II genannten Version, illustriert mit anschaulichen Beispiel von Programmierungswiese und Programmbauweise, umfasst.

Lewoc J. B.: Quo vadis, ESER?

INFORMATYKA 1984, Nr. 3, S. 5

Kritische Bemerkungen zur Trema der Effektivität von R-32 Standartsoftware, die an einem Beispiel von Forschungsergebnissen der bestehenden Steuersystemelösungen für Ferndatenverarbeitung und Mehrfachzugriff begründet wurden. Im Schlussteil stellt der Autor Beschlüsse und Vorschläge zur Verbesserung der Situation vor.

Mirkowski J., Piątkowski A.: Didaktisches Laboratorium im CAMAC System mit verteilter Intelligenz

INFORMATYKA 1984, Nr. 3, S. 11

Eine Charakteristik der auf CAMAC System gebauten Laborlösung zur Durchführung der wissenschaftlichen und didaktischen Experimente sowie Automatisierung der Studentenwerkstätte. Es wurden Hard- und Software, sowie Musterrealisation eines Laboratoriums, das im didaktischen Prozess des Warschauer Technischen Universität eingeführt wurde.

Gacek R., Gościński A., Rempała Z., Rygał R.: ATA-77 Analysator im CAMAC System zur Zweiparameteranalyse

INFORMATYKA 1984, Nr. 3, S. 21

Eine Charakteristik des ATA-77 Analysators, der für Experimentenautomatisierung in Hochspannungslabor vorgesehen wurde. Es wurden Hardware, Wirkungsweise und Software des Systems, sowie seine Vor- und Nachteile besprochen.

Sobczyk M., Szałas A.: PROLOG (1)

INFORMATYKA 1984, No. 3, p. 1

First part of PROLOG language characteristics, which includes concise history of its creation and development, as well as discussion of the PROLOG II improved version, illustrated with a demonstrative example of programming method and program's structure.

Lewoc J. B.. Quo vadis, Unified System?

INFORMATYKA 1984, No. 3, p. 5

Critical remarks of effectiveness of R-32 standard software, based on research results of existing solution for teleprocessing and multiaccess control systems. In the termination the author presents conclusions and proposals for improving this situation.

Mirkowski J., Piątkowski A.: Didactic laboratory in CAMAC system with distributed intelligence

INFORMATYKA 1984, No. 3, p. 11

Characteristics of a laboratory solution realized in CAMAC system for scientific and didactic experiments, as well as for student's workshop automation. Hardware and software, as well as exemplary laboratory realization, introduced to didactic process in the Warsaw Technical University are discussed.

Gacek R., Gościński A., Rempała Z., Rygał R.: ATA-77 analyser in CAMAC system for two-parameter analysis

INFORMATYKA 1984, No. 3, p. 21

Characteristics of the ATA-77 analyser, devoted for experiments automation in high-voltage laboratories. Hardware, operation method and software of the system, as well as its advantages and disadvantages, are discussed.

Komputer – rzecz prywatna

Rozwój technologii układów wysokiej skali integracji spowodował w ostatnich dziesięciu latach lawinę różnego rodzaju urządzeń opartych na mikroprocesorach. W połowie lat siedemdziesiątych pojawiły się kalkulatory elektroniczne; w ciągu paru zaledwie lat uczyniły one suwaki logarytmiczne oraz liczydła eskponatami z muzeum. Wprawdzie w Polsce jeszcze do niedawna można było kupić suwaki produkowane przez spółdzielnię SKALA, ale to już chyba tylko przyzwyczajenie do tradycji, a nie rachunek ekonomiczny. Cena suwaka jest podobna do ceny wielodziałaniowego kalkulatora, a wygody obliczeń i ich dokładności nie można porównywać bez uśmiechu.

Lata osiemdziesiąte to rewolucja mikrokomputerów. Te wyroby niczego nie zastępują ani nie imitują. Niewątpliwie rację ma prof. Schaff przewidując degradację krajów drugiego świata, jeżeli nie wezmą one udziału w wyścigu zastosowań mikroprocesorów. Degradację do czwartego świata, do narodów pozbawionych dostępu do informacji i technologii, słowem — wyłączonych z życia międzynarodowego. Czym więc jest urządzenie, które niesie rewolucję? Uważam, że nie jest nim sam mikroprocesor. Jest nim dopiero k o m p u t e r o s o b i s t y.

Próba zdefiniowania komputera osobistego niesie w sobie pewne niebezpieczeństwo: rozwój technologii oraz fantazja projektantów wyprzedzają rzeczywiste potrzeby użytkowników. Jeszcze nie nauczyliśmy się korzystać w skali masowej z własnego ośrodka informatycznego, jakim jest mikrokomputer, a już dano nam nowe możliwości (np. arcywygodne procesory tekstu).

Do definicji komputera osobistego należałoby zatem włączyć wszystkie urządzenia oparte na mikroprocesorze, a służące j e d n e j osobie. Służące w każdej chwili. A pamiętajmy, że komputer przeznaczony dla wielu osób, może być przez nie użytkowany w innym czasie — tak dzieje się w szkołach i na uniwersytetach. W pewnym sensie mamy tu więc powrót do komputerów wczesnych lat sześćdziesiątych, gdy jeden użytkownik pracował sam na sam z maszyną. Tyle że wtedy maszyny o pojedynczym dostępie zajmowały cały pokój i wymagały wielu osób obsługi. Dziś mikrokomputer stoi na biurku i czeka na wciśnięcie przełącznika „sieć".

Osobisty to też taki, którego konfiguracja zależy tylko od fantazji użytkownika — także od jego zasobów finansowych. Absurdalnym, ale jak najbardziej realistycznym przykładem jest możliwość przyłączenia dysku typu Winchester o pojemności 5 MB do komputera kieszonkowego HP41 o pojemności pamięci ok. 2 KB — wszystkie dzięki uniwersalnemu sprzęgowi HP-IL, który przekształca programowany kalkulator w rzeczywisty komputer. Chociaż przykład ten nie jest poważny, to przecież nigdy nie możemy być pewni, czy ktoś gdzieś nie będzie potrzebował taniego sterownika o prostym układzie pomiarowym oraz bardzo dużej bazy danych. A wtedy absurdalna konfiguracja może stać się rzeczywistością — wykreowaną przez fantazję użytkownika.

Osobisty to też taki, który wszędzie towarzyszy użytkownikowi (jak zegarek, ale nie jak samochód...), czyli prawdziwie osobisty komputer musi być mały i zasilany z akumulatorów. Przenośne powinny być też jego urządzenia zewnętrzne. Wszystko po to, by stale człowiekowi towarzysząc — pozwalały rozwiązywać najrozmaitsze problemy: od obliczania budżetu domowego do skomplikowanych całek, od przypominania o ważnych spotkaniach do optymalizacji zużycia benzyny przez własny samochód. Zresztą, jak dowodzi praktyka, inwencja posiadaczy komputerów osobistych jest niewyczerpana. Świadczy o tym liczba programów, które można już obecnie kupić w wyspecjalizowanych domach handlowych — katalogi takich placówek zawierają tysiące propozycji, mogących zaspokoić najbardziej wyrafinowane gusta. Przykład, który ciśnie się na język: istnieje cała seria programów uczących przyrządzania wyrafinowanych dań kuchni francuskiej. Szkoda, że komputer nie zajmuje się też zaopatrzeniem w produkty wyjściowe...

Ostatni, aczkolwiek chyba najważniejszy warunek komputera osobistego: cel w jakim został on zakupiony. Jeżeli posiadacz komputera używa go w pracy i tylko w tym celu, to komputer taki jest jedynie narzędziem. Natomiast zastosowanie tej samej maszyny do obliczania bilansu wypłat przedsiębiorstwa oraz do gry towarzyskiej — łączy w jedno dwie różne funkcje; staje się odbiciem osobowości użytkownika, jego potrzeb i zainteresowań.

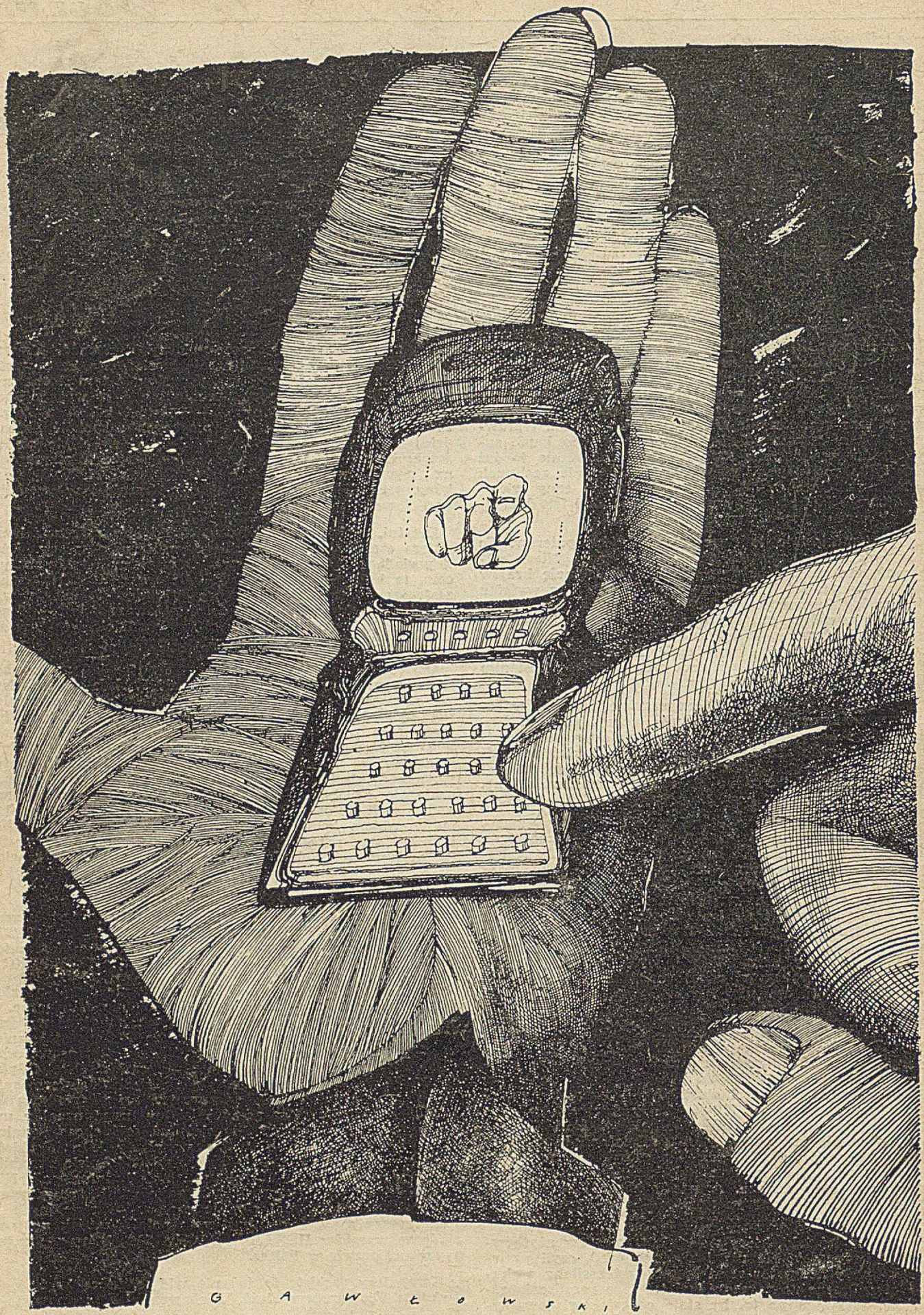
Czy dzisiaj są już dostępne na świecie komputery będące w powyższym sensie osobistymi? Wydaje się, że właśnie rozpoczęła się era takich urządzeń. Co najmniej pięć firm (EPSON, CASIO, TANDY, TI i HP) produkuje już komputery przenośne, wyposażone w język BASIC oraz możliwość sprzęgania z urządzeniami zewnętrznymi. Dzięki postępowi w konstrukcji mechanizmów (drukarki, plotery, napędy dyskietek), cena urządzeń towarzyszących znacznie się obniżyła. To z kolei spowodowało masowe zakupy, czyli dalsze obniżenie cen. Dziś można już nabyć pełny system komputerowy o pamięci operacyjnj 32 KB i podwójnym napędzie dyskietek, z drukarką o szerokości 80 znaków — za średnią pensję miesięczną. Mamy więc do czynienia z urządzeniem masowym (jak samochód i kolorowy telewizor).

A w Polsce? Jak wiemy z artykułu „Komputery zagrażają!" Janusza Gwiazdy (INFORMATYKA nr 10/83), wprawdzie nikt specjalnie nie broni przywozu komputerów, ale...

Po pierwsze — cena. K a ż d e przeliczenie dewiz na złotówki daje sumę astronomiczną. To już nie miesięczne, a wieloletnie zarobki. A przy tym i tak rynek krajowy jest w gruncie rzeczy nasycony. To prawda — ilu, ostatecznie, ludzi w Polsce potrzebuje i ma możliwości (finansowe oraz techniczne) zastosowań komputerów osobistych? Gdyby ktoś zechciał dziś przyłączyć się do sieci teleinformacyjnej poprzez prywatny mikrokomputer, mógłby to uczynić tylko... za granicą. Czy odpowiedzialny urząd uznałby rację wymiany informacji? Przecież już dziś chcąc wymieniać programy z zagranicą, gdy wysyłamy je na dyskietkach płacimy (lub nie — to zależy od widzimisię urzędnika pocztowego) olbrzymie cło za... nośnik informacji. Nic natomiast za sam program. Nie została też rozwiązana sprawa wysyłki komputerów do napraw gwarancyjnych, a nie wszyscy są w tak szczęśliwym położeniu, jak np. posiadacze wyrobów HEWLETT-PACKARDA, który ma autoryzowany serwis w Polsce.

Komputery osobiste to już nie widmo, które krąży nad światem. To rzeczywistość. Niebawem ta rzeczywistość zawita do nas.

JAKUB TATARKIEWICZ


G A W L O W S K I