5 1987

informatyka

Prof. Robert Kowalski o systemach ekspertowych

Synchronizacja transakcji w SRBD

Concurrent DOS

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

5 1987

Nr 5

Miesięcznik Rok XXII

Maj 1987

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

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

Zakł. Graf. „Tamka". Zam. 0134-1300/87. Obj. 4,0 ark. druk. Nakład 8050 egz. K-80.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

WYDAWNICTWO NACZELNA ORGANIZACJA TECHNICZNA
SIGMA
CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Jan Madey omawia problematykę systemów operacyjnych na przykładzie systemu Unix
- Zbigniew Szkaradnik i Wojciech Petrykowski o grafice żółwia w języku Forth
- Czesław Ratka o adaptacji drukarki wierszowej DW 204-2 do pracy z mikrokomputerem SM-4
- Mariusz Kuc o kompilatorze Ady dla mikrokomputerów
- Marek Machura o systemach ekspertowych na mikrokomputery IBM PC

ROBERT KOWALSKI
Imperial College of Science and Technology
Uniwersytet Londyński
Wielka Brytania

Logika w systemach ekspertowych (I)

Pojęcie systemu ekspertowego nie jest zbyt dobrze określone. Niektóre definicje utożsamiają systemy ekspertowe w całości z systemami regułowymi, inne — z szeroko rozumianą dziedziną sztucznej inteligencji [7]. W poniższym artykule przedstawiono argumenty przeciw obu tym punktom widzenia, odwołując się do przykładów w postaci programów w Prologu, zbudowanych z klauzul Horna. Przeanalizowano również słuszność często głoszonej opinii, że realizatorzy systemów ekspertowych mają poważne trudności z zapewnieniem niezbędnej poprawności tworzonego oprogramowania, przytoczono argumenty na to, że zastosowanie logiki przyczynia się istotnie do przezwyciężenia tych problemów.

PROGRAMOWANIE LOGICZNE

W swej najprostszej postaci **program logiczny** jest zbiorem asercji i reguł:

A **jeśli** B **i** C **i** ...

gdzie każdy wniosek A i każdy warunek B, C, ... jest **asercją**, czyli stwierdzeniem istnienia **relacji** między poszczególnymi obiektami danej dziedziny. Tak więc każdy program logiczny jest programem regułowym.

Przykład 1:

Posortowane (x y)
jeśli Permutacja (x y)
i Uporządkowane (y)

Reguła ta stwierdza, że dla wszystkich x i y:

ciąg y jest posortowaną wersją ciągu x, jeśli y jest permutacją x, i jeśli y jest uporządkowane.

Proceduralna reprezentacja reguł, będąca istotą programowania logicznego, traktuje regułę jako procedurę:

aby posortować ciąg x i otrzymać posortowany ciąg y, wyznacz permutację y utworzoną z x i wykaż, że y jest uporządkowane.

Ogólnie, reguły postaci: A **jeśli** B **i** C **i** ...

są interpretowane jako procedury, które redukują zadania postaci A do podzadań B i C i ...

Teoretycznie podzadania mogą być rozwiązywane w dowolnej kolejności, a nawet równolegle. Również reguły mogą być sprawdzane szeregowo lub równolegle. W Prologu rozwiązuje się podzadania i sprawdza reguły w kolejności ich wypisania. W innych językach programowania logicznego, takich jak Parlog [1] i współbieżny Prolog [6], podzadania i reguły traktuje się równolegle. Głównym celem japońskiego projektu piątej generacji [2] jest praktyczne wykorzystanie równoległości programowania logicznego przede wszystkim w takich dziedzinach zastosowań sztucznej inteligencji jak budowa systemów ekspertowych.

SYSTEMY EKSPERTOWE A SYSTEMY REGUŁOWE

W przykładzie 1 zdefiniowano relację sortowania na najwyższym poziomie ogólności. Nie jest ona oczywiście zalążkiem systemu ekspertowego, ani nawet programem w sensie konwencjonalnym. Jej napisanie w postaci reguły jest jednocześnie przykładem podważającym zasadność utożsamiania systemów ekspertowych z systemami regułowymi.

Z drugiej jednak strony istnieją bliskie związki między systemami ekspertowymi a systemami regułowymi. Praktycznie każdy system ekspertowy o znacznym poziomie złożoności jest zbudowany jako system regułowy; i odwrotnie, w Ameryce Północnej systemy ekspertowe były do tej pory prawie wyłączną domeną języków regułowych. W Europie natomiast Prolog od dawna był używany jako język regułowy w wielu różnych dziedzinach zastosowań, takich jak bazy danych, przetwarzanie języka naturalnego, projektowanie wspomagane komputerowo, pisanie kompilatorów i szybkie konstruowanie prototypów. Wiele spośród tych zastosowań zostałoby sklasyfikowanych (słusznie czy też nie) jako systemy ekspertowe, gdyby zrealizowano je w Ameryce Północnej.

Przykład 2:

Posortowane (x x)
jeśli Uporządkowane (x)
Posortowane (x y)
jeśli $k < l$
i $x_k > x_l$
i Wymienione (x k l z)
i Posortowane (z y)

Te dwie reguły tworzą najwyższy poziom algorytmu sortującego ciągi x:
jeśli x jest uporządkowane, to x jest posortowane. W przeciwnym razie, aby posortować x wybierz nieuporządkowa-

Artykuł jest poprawioną wersją wykładu przygotowanego przez autora na seminarium poświęcone komputerom piątej generacji, które odbyło się podczas posiedzenia Stowarzyszenia Europejskich Użytkowników Komputerów Control Data (ECODU) w Londynie, w dniu 21 kwietnia 1986 roku.

Prof. ROBERT KOWALSKI studiował na Uniwersytecie w Bridgeport (stopień B.A. z matematyki w 1963 r.), Uniwersytecie Stanfordzkim (stopień magistra nauk matematycznych w 1966 r.) oraz Uniwersytecie Edynburskim (tytuł doktora nauk informatycznych w 1970 r.).

W latach 1967—1975 był pracownikiem naukowym na Wydziale Logiki Obliczeniowej Uniwersytetu Edynburskiego. W roku 1975 przeniósł się do Imperial College of Science and Technology w Londynie, gdzie objął stanowisko wykładowcy w dziedzinie teorii obliczeń. W październiku 1982 r. otrzymał nominację na stanowisko profesora w dziedzinie logiki obliczeniowej.

Pierwsze prace badawcze R. Kowalskiego dotyczyły zagadnień automatycznej dedukcji. W wyniku tych badań oraz współpracy z Alainem Colmerauerem z Uniwersytetu w Aix-Marseille zostały stworzone podstawy do wykorzystania logiki formalnej jako języka programowania. Prace R. Kowalskiego zapewniły podbudowę teoretyczną, natomiast prace A. Colmerauera doprowadziły do zdefiniowania Prologu, języka programowania logicznego.

Prof. R. Kowalski kieruje obecnie Grupą Programowania Logicznego w Imperial College. Grupa ta zajmuje się realizacją i zastosowaniami Prologu oraz Parlogu, równoległego języka programowania w logice. Podstawowe zastosowania obejmują dziedzinę systemów ekspertowych oraz zagadnienia związane z formalizacją ustawodawstwa.

ną parę x_k, x_l, wymień jej elementy i posortuj nowo powstały ciąg.

Równoważne sformułowanie brzmi:

aby posortować ciąg x powtarzaj wymianę elementów w nieuporządkowanych parach tak długo, aż x zostanie uporządkowane.

Nierówność $x_k > x_l$ jest tutaj pewną formą notacji funkcjonalnej, którą można łatwo przekształcić na postać relacyjną. Wybór indeksów k i l jest niedeterministyczny w tym sensie, że kolejność wyboru nieuporządkowanych par nie ma żadnego znaczenia [4].

Przykładu 2 nie można uznać za system ekspertowy w potocznym rozumieniu. Niemniej jednak jest on opisany za pomocą języka reguł i w porównaniu z przykładem 1 zawiera w sobie pewną wiedzę o sortowaniu ciągów. Pobieżne przejrzenie trzeciego tomu monografii Knutha [3] daje wyobrażenie o rozmiarach wiedzy dostępnej w dziedzinie sortowania.

GRA W PIĘTNAŚCIE

Być może słabą stroną przykładu sortowania jako systemu ekspertowego jest jego algorytmiczny charakter. Warto więc przytoczyć inny przykład z dziedziny sztucznej inteligencji, który lepiej obrazuje różnicę między systemem „naiwnym" a systemem ekspertowym. Przykładem tym, często cytowanym w literaturze, jest gra w piętnaście. W artykule [5] opisano adaptacyjny program rozwiązujący prostszą wersję tego problemu dla ośmiu tabliczek.
Gra w piętnaście polega na wygenerowaniu ze stanu początkowego, np.:

2	10	6	5
13		3	12
1	14	9	8
4	15	11	7

stanu docelowego

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

przez wyznaczenie odpowiedniego ciągu dopuszczalnych ruchów. Ruch dopuszczalny polega na wymianie pustego miejsca z najbliższą — w pionie lub poziomie — tabliczką.

Klasyczne rozwiązanie tego zadania można przedstawić za pomocą następującej definicji regułowej:

Posortowane (x x)
jeśli Uporządkowane (x)
Posortowane (x y)
jeśli x_k = puste
i k bliskie l
i Wymienione (x k l z)
i Posortowane (z y)

Pomimo powierzchniowego podobieństwa do omawianej poprzednio definicji sortowania, definicja ta przypomina bardziej specyfikację programu niż sam program. Interpretując tę definicję proceduralnie otrzymuje się przestrzeń rozwiązań złożoną z kolejnych alternatywnych ruchów dopuszczalnych. W Prologu można skoncentrować się w danej chwili tylko na jednej gałęzi przestrzeni rozwiązań, stosując strategię przeszukiwania w głąb z nawrotami.

Nieefektywność tej strategii jest przyczyną nieprzydatności Prologu w dowodzeniu twierdzeń. Strategia przeszukiwania w głąb jest jednak jądrem każdego procesora Prologu. Uważa się, że charakterystyczną cechą Prologu jest właśnie zapewnienie **delikatnej równowagi** między dowodzeniem twierdzeń a wykonywaniem programów.

Klasyczne metody sztucznej inteligencji pokonują ograniczenia strategii przyjętej w Prologu przez uwzględnienie zarówno opisu zadania, jak i związanej z nim przestrzeni rozwiązań. Znalazło to wyraz w bogatej literaturze poświęconej przeszukiwaniu heurystycznemu i ogólnym, niezależnym dziedzinowo strategiom rozwiązywania zadań.

Podejście przyjęte w systemach ekspertowych polega na zastąpieniu ogólnej wiedzy o rozwiązywaniu zadań przez wiedzę szczegółową z danej dziedziny zastosowań. Zadanie gry w piętnaście ekspert mógłby rozwiązać wykonując na przykład następującą sekwencję kroków.

Krok 1
Uporządkuj pierwszy rząd.

1	2	3	4
	6	13	5
10	9	8	12
15	14	11	7

Krok 2
Uporządkuj drugi rząd, nie przesuwając tabliczek w pierwszym rzędzie.

1	2	3	4
5	6	7	8
13	10	12	
15	14	11	9

Krok 3a
Uporządkuj pierwszą kolumnę w rzędach 3 i 4, nie przesuwając tabliczek w rzędach 1 i 2.

1	2	3	4
5	6	7	8
9		14	11
13	15	10	12

Krok 3b
Uporządkuj drugą kolumnę w rzędach 3 i 4, nie przesuwając tabliczek w rzędach 1 i 2, ani w kolumnie 1 w rzędach 3 i 4.

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

Krok 3c
Uporządkuj trzy pozostałe tabliczki, nie przesuwając innych tabliczek.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Podobnie każdy z powyższych kroków można rozłożyć na sekwencję mniejszych kroków, wykorzystując szczegółową wiedzę o rozwiązywaniu podzadań.

Do opisania tego procesu za pomocą reguł posłużą dwie reguły podstawowe:

- Posortowany-pierwszy-wiersz (u v u' v')
stwierdzająca, że jeśli u jest posortowane, a v nie jest posortowane, to sortując pierwszy wiersz v bez ingerencji w u, otrzymuje się posortowane u' i nieposortowane v';
- Posortowana-pierwsza-kolumna (u v u' v')
stwierdzająca, że jeśli u jest posortowane, a v nie jest posortowane, to sortując pierwszą kolumnę v bez ingerencji w u, otrzymuje się posortowane u' i nieposortowane v'.

Niech () oznacza stan pusty. Proceduralny opis kroków eksperta ma następującą postać:

Posortowane (x y)
jeśli Posortowany-pierwszy-wiersz (() x y_1 x_1)
i Posortowany-pierwszy-wiersz (y_1 x_1 y_2 x_2)
i Posortowana-pierwsza-kolumna (y_2 x_2 y_3 x_3)
i Posortowana-pierwsza-kolumna (y_3 x_3 y_4 x_4)
i Posortowana-pierwsza-kolumna (y_4 x_4 y_5 x_5)
i Posortowana-pierwsza-kolumna (y_5 x_5 y ())

Dla uproszczenia opisu krok 3c zastąpiono dwoma krokami.

Rozwiązanie zadania gry w piętnaście techniką systemów ekspertowych ma ciągle zbyt algorytmiczny charakter. Przykład ten ilustruje znaczenie wiedzy szczegółowej w danej dziedzinie zastosowań w porównaniu z rolą wiedzy ogólnej o metodach rozwiązywania zadań, nie ukazuje natomiast heurystycznego, empirycznego stylu rozwiązywania skomplikowanych zadań, typowego dla systemów ekspertowych. Cechę tę można zobrazować na przykładzie rozwiązywania równań matematycznych. Zanim jednak skoncentrujemy się na samym systemie ekspertowym, przyjrzyjmy się bliżej specyfikacji zadania.

ROZWIĄZYWANIE RÓWNAŃ PRZY UŻYCIU REGUŁ ALGEBRY

Rozważmy bardzo proste równanie:
(2*t) = s

które należy rozwiązać ze względu na t. Zastosowanie reguł algebry można zilustrować za pomocą następującego obliczenia:

Dane
(2*t) = s

Krok 1
Zamień (2*t) na (t*2).
(t*2) = s

Krok 2
Podziel obie strony równania przez 2.
((t*2)/2) = (s/2)

Krok 3
Zamień ((t*2) /2) na (t*(2/2)).
(t*(2/2)) = (s/2)

Krok 4
Zamień (2/2) na 1.
(t*1) = (s/2)

Krok 5
Zamień (t*1) na t.
t = (s/2)

Cel
Każdy krok (z wyjątkiem kroku 2) opiera się na ogólnej regule zamiany:
x = y
jeśli x' = y
i u w wyrażeniu x'
i u = v
i zamiana u na v w wyrażeniu x' daje x.

Reguła ta łącznie z regułą

x = y **jeśli** y = x

umożliwia zastąpienie podwyrażenia u podwyrażeniem v po obu stronach równości. Obie relacje „w wyrażeniu" i „zamiana" mogą być w prosty sposób zdefiniowane za pomocą klauzul Horna. Wyrażenia postaci (operand1 operator operand2) można przedstawić w notacji Prologu jako listy trójelementowe. Pozostałe reguły niezbędne do wykonania przekształceń zawartych w krokach 1—5 mają postać:

Krok 1
(x*y) = (y*x)

Krok 2
(x/z) = (y/z) jeśli x = y i $z \neq 0$

Krok 3
(x/y) = (x*(y wykładnik —1))
((x*y)*z) = (x*(y*z))

Krok 4
(x/x) = 1 jeśli $x \neq 0$

Krok 5
(x*1) = x

Dla uproszczenia zbioru reguł dzielenie przez y wyrażono jako mnożenie przez odwrotność y.

Rozwiązywanie równań z wykorzystaniem reguł algebry przypomina w istocie zadanie dowodzenia twierdzeń. Chociaż reguły można wyrazić za pomocą klauzul Horna, w Prologu przebiega się je według strategii wnioskowania zstępującego, sprawdza według strategii przeszukiwania w głąb i natychmiast „wpada" w nieskończoną pętlę.

W klasycznych metodach sztucznej inteligencji zastosowano by tutaj raczej inne podejście. Reguły byłyby przebiegane według strategii wnioskowania wstępującego z odpowiednim mechanizmem przeszukiwania, opartym na wykrywaniu nieskończonych pętli. Jednakże również i w tym wypadku zadanie nasze byłoby narażone na niebezpieczeństwo „kombinatoryjnej eksplozji".

Z punktu widzenia inżynierii oprogramowania reguły algebry są pewną specyfikacją. Inżynierowie oprogramowania nie próbują nawet wykonywać specyfikacji; przekształcają je na wykonywalne programy. Stosując klasyczne podejście do projektowania programów, inżynier oprogramowania zmierzałby w naszym przykładzie do skonstruowania jednego algorytmu, rozwiązującego wszystkie równania z danej klasy równań. W podejściu charakterystycznym dla systemów ekspertowych „inżynier wiedzy" konstruuje zbiór reguł przyrostowo, zwiększając stopniowo zakres rozwiązywania problemów. O ile inżynier oprogramowania stara się od razu stworzyć projekt docelowego programu, inżynier wiedzy buduje zbiór reguł metodą prób i błędów.

Opracował i tłumaczył
MAREK MACHURA

LITERATURA

[1] Clark K. L., Gregory S.: PARLOG — a parallel logic programming language. Report DOC 83/5, Imperial College, Londyn, 1983
[2] Fuchi K.: The direction the FGCS project will take. New Generation Computing. Vol. 1, No. 1, 1983
[3] Knuth D. E.: The Art of Computer Programming. Vol. 3, Addison-Wesley, Reading (MA), 1973
[4] Kowalski R. A.: Logic Programming. Invited Paper, Proceedings of IFIP Congress, Paris, 1983
[5] Michie D., Ross R.: Experiments with the Adaptive Graph Traverser. Machine Intelligence. Vol. 5, pp. 301—18, 1969
[6] Shapiro E., Takeuchi A.: Object Oriented Programming in Concurrent PROLOG. New Generation Computing. Vol. 1, No. 1, 1983
[7] Stefik A. et al.: The organization of expert systems, a tutorial. Artificial Intelligence. Vol. 18, No. 2, March 1982.

JAN WĘGLARZ
Instytut Automatyki
Politechnika Poznańska

POLSKIE TOWARZYSTWO INFORMATYCZNE

Ocena działania systemów komputerowych — przedmiot, metody, kierunki rozwoju (2)

W drugiej części artykułu omówiono modelowanie obciążenia i przedstawiono niektóre kierunki rozwoju metod oceny działania systemów komputerowych.

MODELOWANIE OBCIĄŻENIA

Właściwy model obciążenia systemu ma podstawowe znaczenie w badaniach ewaluacyjnych, ze względu na to, że w badaniach tych prawie wyłącznie posługujemy się modelami rzeczywistego obciążenia. Tak jest zawsze w wypadku metod modelowania (analitycznych i symulacyjnych), a na ogół także w wypadku metod pomiarowych, gdy wykorzystuje się próbki rzeczywistego obciążenia, które — trzeba to wyraźnie podkreślić — także stanowią jego model.

Przed przystąpieniem do konstruowania modelu obciążenia należy sprecyzować poglądy w dwóch sprawach. Pierwsza dotyczy granic systemu, od których zależy skład obciążenia. Zwykle zadania (procesy) użytkowników są traktowane jako zewnętrzne w stosunku do systemu, a zatem wchodzące w skład obciążenia, natomiast moduły systemu operacyjnego zarządzające zasobami są zaliczane do systemu ocenianego. Te zaś programy systemowe, które są wykonywane na żądanie indywidualnego użytkownika (kompilatory, asemblery, edytory itp.), tworzą „strefę graniczną", a ich zaliczenie do systemu lub obciążenia zależy od rozpatrywanego problemu ewaluacyjnego. Druga sprawa dotyczy typu i rodzaju obciążenia, które będzie modelowane (np. obliczenia naukowo-techniczne, handlowe, warunki normalne, szczytowe itp.) oraz wymaganego horyzontu czasowego obciążenia (godzinowe, dobowe, tygodniowe, roczne). Trzeba pamiętać, że wraz ze zmniejszaniem się horyzontu czasowego pojawia się problem uwzględnienia wpływu warunków granicznych.

Ważnym zagadnieniem jest także ocena i wybór modelu obciążenia, jako że na ogół nie jest on jednoznacznie narzucony przez specyfikę problemu ewaluacyjnego. Ocenę można wykonać z wielu punktów widzenia, z których najważniejsze to:

- reprezentatywność, tj. adekwatność, w sensie wartości badanych kryteriów oceny działania, do modelowanego obciążenia rzeczywistego;
- elastyczność, tj. podatność na modyfikacje odwzorowujące zmiany w modelowanym obciążeniu rzeczywistym;
- koszt konstruowania, tj. koszt zbierania informacji niezbędnej do budowy modelu;
- koszt wykorzystania;
- zwartość (w odniesieniu do określonej reprezentatywności i kosztu);
- niezależność od systemu, tj. możliwość przeniesienia do innego (także w sensie wprowadzonych modyfikacji) systemu bez utraty reprezentatywności;
- powtarzalność;
- kompatybilność z systemem lub jego modelem.

Waga powyższych kryteriów zależy oczywiście od typu problemu ewaluacyjnego i metody jego rozwiązywania, na przykład, niezależność od systemu jest szczególnie ważna w problemach wyboru, a powtarzalność odgrywa istotną rolę w wypadku metod pomiarowych.

Poniżej krótko przedstawiono główne typy modeli obciążenia.

Modele naturalne

Modele naturalne są to próbki rzeczywistego obciążenia, pod którym system aktualnie się znajduje. Nie zalicza się zatem do modeli naturalnych próbek obciążenia zapamiętanych i później podanych na wejście systemu. Konstruowanie tych modeli sprowadza się więc do określenia momentów rozpoczęcia i zakończenia eksperymentów pomiarowych. Należy przy tym zwrócić baczną uwagę na reprezentatywność modelu, a więc na stacjonarność istotnych parametrów obciążenia i na zgodność uzyskiwanych z pomiarów estymatorów badanych kryteriów oceny, czyli ich stochastyczną zbieżność do wartości tych kryteriów przy rzeczywistym obciążeniu. Przed przystąpieniem do wykorzystania modeli naturalnych należy sobie zatem przypomnieć podstawowe wiadomości z teorii estymacji. Po uwzględnieniu tych warunków można uzyskać dobrą reprezentatywność modelu.

Ocena modeli naturalnych z pozostałych punktów widzenia jest następująca: elastyczność — bardzo mała, koszt konstruowania — mały, koszt wykorzystania — dość duży (długie sesje pomiarowe), zwartość — zwykle mała (z uwagi na wymaganą reprezentatywność), niezależność od systemu — mała (nawet niewielkie modyfikacje systemu mogą spowodować niezgodność estymatorów), powtarzalność — bardzo mała, kompatybilność — pełna. Główną zaletą tych modeli jest to, że nie wymagają one przerwania normalnej pracy systemu.

Wykonywalne modele sztuczne

Wykonywalne modele sztuczne są to modele złożone z programów wykonywalnych w danym systemie komputerowym, a nie będące modelami naturalnymi. Wykorzystuje się je głównie w metodach pomiarowych (z wyjątkiem symulatorów sterowanych programami wykonalnymi w danym systemie) w sytuacjach, gdy modele naturalne są nieprzydatne lub uciążliwe w zastosowaniu. Pierwsza sytuacja występuje, na przykład, przy porównaniu kilku systemów (problemy wyboru) lub przy badaniu działania systemu pod obciążeniem przewidywanym w przyszłości; druga — na przykład wtedy, gdy trzeba zbadać wpływ nie w pełni sprawdzonych modyfikacji systemu operacyjnego na działanie systemu ocenianego. Zastosowanie modelu naturalnego narażałoby wówczas użytkowników systemu na trudne do zinterpretowania błędy, a ponadto miałoby niewielki zakres z uwagi na małą powtarzalność tego modelu.

Do najdłużej stosowanych wykonywalnych modeli sztucznych należą tzw. mieszanki, w szczególności, mieszanka rozkazowa, będąca zbiorem względnych częstości wykonywania głównych klas rozkazów, wspólnych dla szerokiej klasy procesorów. Oczywiście, mieszanka jako taka nie jest wykonywalna, ale każdy program charakteryzujący się takimi samymi względnymi częstościami wykonywania odpowiednich rozkazów jest już modelem wykonywalnym. Mieszanka rozkazowa uzyskana z pomiarów w systemie o typowej architekturze i dla typowego obciążenia (np. obliczeń naukowo-technicznych) może być wykorzystywana także do badań ewaluacyjnych w podobnych systemach pod zbliżonym obciążeniem. Do takich standardowych mieszanek rozkazowych należą: mieszanka Gibsona — dla systemu IBM 7090 oraz mieszanka Flynna — dla systemu IBM 360 [8] (tabela).

Mieszanki Gibsona i Flynna

Klasa rozkazów	Częstość [%]	
	Gibson	Flynn
Ładowanie pamięci	31,2	45,1
Indeksowanie	18	
Skok warunkowy	16,6	27,5
Porównanie	3,8	10,8
Arytmetyka stałoprzecinkowa	6,9	7,6
Arytmetyka zmiennoprzecinkowa	12,2	3,2
Przesunięcie logiczne	6	4,5
Inne	5,3	1,3
	100,0	100,0

Mieszanki mogą odpowiadać różnym typom i rodzajom obciążenia oraz dotyczyć oddzielnie kompilacji i wykonywania programów. Mogą one być tworzone także na poziomie wyższym niż poziom języka wewnętrznego; na poziomie języków takich jak Fortran czy Algol stosuje się tzw. mieszanki instrukcyjne, na poziomie języków poleceń operatorskich — mieszanki poleceniowe itd. Ogólną wadą mieszanek jest to, że z samej swej natury nie zawierają żadnej informacji o współzależnościach w modelowanym obciążeniu. Ich skuteczne zastosowanie jest zatem ograniczone do tych problemów ewaluacyjnych i architektur systemów, dla których uzasadnione jest założenie, że wpływ na zachowanie się systemu mają pojedyncze rozkazy, (instrukcje, polecenia) a nie ich ciągi. W szczególności, założenia takiego nie można przyjąć w systemach z procesorami wykorzystującymi zakładkowanie. Po uwzględnieniu tej uwagi ocena mieszanek jest następująca: reprezentatywność — średnia, elastyczność — duża, koszt konstruowania — średni, koszt wykorzystania — mały, zwartość — duża, niezależność od systemu — dość duża, powtarzalność — pełna, kompatybilność — pełna.

Innym typem wykonywalnych modeli sztucznych są wykonywalne ślady. Zależą one od rodzaju badanego systemu. Dla systemu z przetwarzaniem wsadowym ślad składa się z chronologicznie uporządkowanych kart sterujących, dla systemu interakcyjnego zawiera polecenia użytkowników i czasy zastanawiania się użytkowników (tj. czasy od momentu zakończenia przetwarzania poprzedniego polecenia do momentu zakończenia wprowadzania danych następnego polecenia).

Wykonywalne ślady nie mają głównej wady mieszanek, gdyż odwzorowują współzależności w modelowanym obciążeniu. Także ich reprezentatywność może być lepsza przy odpowiednim doborze długości śladu i uwzględnieniu wpływu procedury zdejmowania śladu na pamiętane czasy zdarzeń. Ten ostatni czynnik oraz konieczność starannego inicjowania systemu dla zapewnienia powtarzalności eksperymentów (np. wszystkie pliki związane z zapamiętanymi zadaniami muszą być zapamiętane i ponownie zapisane w pamięci w miejscach, które zajmowały w czasie zdejmowania śladu) stanowią główne wady wykonywalnych śladów. W porównaniu z mieszankami cechuje je również mniejsza elastyczność, zwartość i niezależność od systemu, a większy koszt wykorzystania.

Trudno jest podać ogólną charakterystykę niewykonywalnych modeli sztucznych, typowych dla metod modelowania, gdyż jest ich bardzo wiele i są bardzo różnorodne, począwszy od jednej lub dwóch zdeterminowanych wartości (czas procesora, rozmiar rekordu) aż do bardzo szczegółowych śladów dla symulatorów sterowanych śladem.

NIEKTÓRE KIERUNKI ROZWOJU

Jest rzeczą oczywistą, że na kilku stronach nie można choćby krótko scharakteryzować licznych kierunków rozwoju badań ewaluacyjnych. Ścisłe określenie wielu z nich wymagałoby wprowadzenia skomplikowanych definicji, znacznie wykraczających poza ramy tego artykułu. Także zacytowanie głównych prac związanych z poszczególnymi kierunkami nie jest możliwe do zrealizowania. Omówimy zatem jedynie tzw. podejścia globalne do poprawy działania systemów scentralizowanych oraz niektóre problemy oceny działania systemów rozproszonych. Te ostatnie wykraczają poza ramy poprzednich punktów, w których była mowa zasadniczo o systemach scentralizowanych, jednak w tym rozdziale winny być zasygnalizowane, jako że już dziś stanowią główny kierunek rozwoju badań ewaluacyjnych.

Do podejść globalnych dla systemów scentralizowanych zalicza się te podejścia, w których rozpatruje się więcej niż jeden rodzaj zasobów systemu komputerowego (do tego samego rodzaju zalicza się wszystkie jednostki zasobu spełniające identyczne funkcje) [3, 21]. Poniżej zwrócono uwagę tylko na aspekty analityczne tych podejść, choć na ogół jest to dopiero podstawa do konstrukcji algorytmów, których wpływ na ocenę działania systemu jest określany na drodze odpowiednio ukierunkowanego eksperymentu symulacyjnego.

Dla metod probabilistycznych najbardziej rozpowszechnionym podejściem globalnym jest to, w którym model systemu stanowi sieć kolejkowa, złożona z **n** wierzchołków, reprezentujących rodzaje zasobów, a więc zawierających określone liczby stanowisk obsługi (jednostek zasobu danego rodzaju). Po uzyskaniu obsługi w wierzchołku **i-tym** zadanie przechodzi do wierzchołka **j-tego** z prawdopodobieństwem p_{ij} lub opuszcza sieć z prawdopodobieństwem $1-\sum_{j=1}^{n} p_{ij}$

Jeśli w sieci znajduje się dokładnie **k** zadań (zewnętrzne dopływy i odpływy zadań są zabronione) oraz $\sum_{j=1}^{n} p_{ij} = 1$ dla każdego **i**, to mówi się o sieci zamkniętej, w przeciwieństwie do otwartej. Często bada się zamkniętą sieć kolejkową ze sprzężeniem zwrotnym do tzw. wierzchołka terminali. Zastosowanie sieci kolejkowych (zwłaszcza zamkniętych) w badaniach ewaluacyjnych dało dobre rezultaty, szczególnie w zakresie wykrywania wąskich gardeł, a więc wierzchołków tzw. nasyconych, przy których dla $k \to \infty$ tworzą się nieskończone kolejki zadań (por. [9], tom II).

Wiąże się to oczywiście z doborem liczb jednostek zasobów poszczególnych rodzajów, zapewniających osiąganie w całym systemie stanu równowagi statystycznej. Jednak szersze wykorzystanie tego podejścia jest ograniczone ze względu na trudności z modelowaniem dowolnych żądań i uwolnień zasobów, co ma istotne znaczenie m.in. dla maksymalizacji stopnia wykorzystania zasobów, zwłaszcza z uwzględnieniem rozwiązania problemów zakleszczenia (ang. deadlock) i stałego zablokowania (ang. permanent blocking, starvation). Także założenie znajomości odpowiednich prawdopodobieństw i rozkładów prawdopodobieństwa, wymagane w tym podejściu, może okazać się mało realistyczne w wielu zastosowaniach.

Wynika stąd dążenie do konstruowania podejść deterministycznych, w sensie zdefiniowanym w pierwszej części artykułu. Wśród tych podejść można wyróżnić dwa nurty. W pierwszym przyjmuje się założenia identyczne jak w klasycznej teorii szeregowania zadań na procesorach, przy czym rozpatruje się także inne zasoby systemu (por. 1, 14). W wypadku zasobów nieprzywłaszczalnych przyjmuje się przy tym, że wszystkie żądania zasobowe zadań są znane a priori i realizowane w całości od momentu rozpoczęcia do momentu zakończenia wykonywania zadań. W ten sposób zapobiega się oczywiście powstaniu zakleszczenia, jednak kosztem niskiego stopnia wykorzystania zasobów. Tak mocne założenia pozwalają jednakże na subtelną analizę złożoności obliczeniowej odpowiednich problemów z określeniem granicy, do której istnieją algorytmy dokładne wielomianowe, a także na rozpoznanie mechanizmu przydziału zasobów, który można wykorzystać do konstruowania algorytmów przybliżonych przy słabszych założeniach. Te ostatnie stanowią właśnie przedmiot zainteresowania drugiego nurtu podejść, z których warto wspomnieć o jednym (a raczej o całej klasie ze wspólną ideą), w którym przyjmowane założenia są najsłabsze [3]. Zakłada się w nim mianowicie znajomość (a priori) jedynie zamówień zasobowych zadań, tzn. maksymalnych liczb jednostek zasobów poszczególnych rodzajów jednocześnie wykorzystywanych przez każde zadanie. Nie zakłada się natomiast znajomości liczb żądanych i uwalnianych jednostek zasobów oraz wystąpień żądań przydziałów i uwolnień zasobów. Momenty gotowości zadań także nie są a priori znane.

Oryginalne metody unikania zakleszczenia oraz wykrywania i likwidacji stałego zablokowania, opracowane przez autora tego podejścia [2, 3], wyznaczają w każdej chwili zbiory zadań, których żądania zasobowe mogą być spełnione. Żądania zadań w tych zbiorach są szeregowane zgodnie z algorytmami priorytetowymi, identycznymi dla zaso-

bów przywłaszczalnych i nieprzywłaszczalnych, przy czym priorytety są nadawane w zależności od kryterium oceny działania systemu, z wykorzystaniem dostępnej wiedzy o charakterystykach zadań. Całe podejście ma strukturę zdarzeniową, wyróżniającą algorytmy obsługi następujących zdarzeń: żądanie przydziału zasobu, uwolnienie zasobu, rozpoczęcie wykorzystywania zasobu, zakończenie wykorzystywania zasobu, redukcja zamówienia zasobowego i wykrycie zadania stale zablokowanego.

W badaniach ewaluacyjnych związanych z systemami rozproszonymi wyróżnia się dwa, szczególnie ważne, ogólne obiekty: sieci komputerowe SK i rozproszone bazy danych RBD. Taki sposób wyróżnienia zakłada, że na poziomie SK nie ma znaczenia aspekt samego zastosowania sieci (w tym wypadku RBD), a na poziomie RBD przyjmuje się, że istnieje SK, spełniająca wszystkie swe funkcje. Jest to oczywiście dekompozycja ogólnego problemu oceny działania systemu.

Dekompozycja ta musi zresztą iść głębiej, wyróżniając w wypadku SK strukturę logiczną (wynikającą ze strukturalizacji funkcji komunikacyjnych), strukturę konfiguracyjną (wynikającą z odwzorowania struktury logicznej na składniki sieci, tj. z implementacji funkcji komunikacyjnych w tych składnikach) i strukturę topologiczną (lokalizacja węzłów, koncentratorów, terminali, zasobów informacyjnych, wybór struktury połączeń). Wybór i ocena tych struktur oraz ich wzajemne związki stanowią prawdziwą kopalnię problemów ewaluacyjnych na etapie projektowania SK. Niektóre z tych problemów zostały naświetlone w [12]. Jeśli chodzi o problemy ulepszania działania SK, to aktualnie prowadzone prace dotyczą głównie podsieci komunikacyjnej i poszczególnych protokołów.

Ostatnio, wraz z rozwojem bardzo szybkich (MB/s) sieci lokalnych, dużego znaczenia nabierają problemy sterowania w czasie rzeczywistym protokołami sieci, w celu adaptacyjnej maksymalizacji wydajności i jakości usług. Ocenę metod pomiarowych w SK pod kątem rozwiązywania tych problemów zawiera praca [6], a próbę zdefiniowania odpowiednich miar oceny działania protokołów — praca [7]. W [13] przedstawiono propozycję metody oceny działania protokołów do celów sterowania nimi w czasie rzeczywistym, wykorzystującą monitorowanie transmisji danych do zbierania danych pomiarowych i modele formalne protokołów do interpretacji tych danych.

Jeśli chodzi o rozproszone bazy danych, to wiele problemów ewaluacyjnych dotyczy wyboru struktury RBD, rozproszenia danych i mocy obliczeniowej systemu, doboru liczby kopii danych fizycznych i optymalizacji bibliotek systemowych. Przegląd tych problemów można znaleźć w [11]. Wiele nowych problemów dotyczy zagadnień zarządzania RBD i koncentruje się na dwóch kierunkach: poprawie efektywności mechanizmów synchronizacji transakcji (por. [5, 10]) oraz optymalizacji wykonywania transakcji (por. [3, 4]).

* * *

Na zakończenie warto podkreślić rolę, jaką w formułowaniu i rozwiązywaniu problemów ewaluacyjnych spełniają badania operacyjne. Można śmiało powiedzieć, że stanowią one podstawę metod analitycznych badań ewaluacyjnych, a w znacznej mierze także metod symulacyjnych. Dlatego dalszy rozwój tych metod wiąże się ściśle z rozwojem badań operacyjnych i z rozwojem przepływu postulatów i idei pomiędzy tymi dziedzinami. Wynika stąd również rola badań operacyjnych w kształceniu informatyków [1].

LITERATURA

[1] Błażewicz J.: Problemy optymalizacji kombinatorycznej — złożoność obliczeniowa, algorytmy optymalizacyjne. PWN, Warszawa (w druku)

[2] Błażewicz J., Cellary W., Słowiński R., Węglarz J.: Badania operacyjne dla informatyków. WNT, Warszawa, 1983

[3] Cellary W.: Rozdział zasobów w systemach komputerowych — próby podejścia globalnego. Rozprawy, nr 127, Wydawnictwa Uczelniane Politechniki Poznańskiej, 1981

[4] Cellary W., Królikowski Z., Morzy T.: Analytical model for performance evaluation of transactions (w druku)

[5] Cellary W., Morzy T.: Bi-ordering approach to concurrency control in distributed database systems (w druku)

[6] Cellary W., Stroiński M.: Analysis of methods of computer networks performance measurement. Bux W., Rudin H. (eds.), Performance of Computer Communication Systems. North-Holland, Amsterdam, 1984

[7] Cellary W., Stroiński M.: Performance parameters of computer network protocols refered to their formal model. Proc. 4th Workshop on Protocol Specification, Testing and Verification, 1984

[8] Ferrari D.: Computer Systems Performance Evaluation. Prentice-Hall, Englewood Cliffs (NJ), 1978

[9] Kleinrock L.: Queuing Systems. Vol. I — Theory, Vol. II — Computer Applications, Wiley, New York, 1975, 1976

[10] Morzy T.: Ordered-transaction approach to performance evaluation of concurrency control algorithms for distributed database systems. Akoka J. (ed.), Management of Distributed Data Processing. North-Holland, Amsterdam, 1982

[11] Lampsen B. W., Paul M., Siegert H. J. (eds.): Distributed Systems — Architecture and Implementation. An Advanced Course. Lecture Notes in Computer Science, No. 105, Springer Verlag, Berlin, 1981

[12] Reiser M.: Performance Evaluation of Data Communication Systems. Proc. of the IEEE, Vol. 70, No. 2, February 1982

[13] Stroiński M.: External measurement method for computer network protocols pefformance evaluation. CMG Transactions, June 1983

[14] Węglarz J.: Multiprocessor scheduling with memory allocation — a deterministic approach. IEEE Trans. on Computers, Vol. C-29, No. 8, 1980

[15] Węglarz J.: Towards global approaches to resource allocation in computer systems. Proc. INFORMATICA'82, 1982.

Konferencje

COLING'88

W dniach od 22 do 27 sierpnia 1988 roku odbędzie się dwunasta Międzynarodowa Konferencja Lingwistyki Obliczeniowej, zorganizowana pod auspicjami Międzynarodowego Komitetu Lingwistyki Obliczeniowej (ICCL) przez Towarzystwo Nauk Komputerowych im. Johna von Neumanna, przy współudziale Instytutu Informatyki i Automatyki oraz Instytutu Lingwistyki Węgierskiej Akademii Nauk.

Tematyka konferencji obejmuje:

- zagadnienia teoretyczne związane z lingwistyką, matematyką, informatyką i naukami poznawczymi (ang. cognitive science);
- modele obliczeniowe (fonemika, morfika, składnia, semantyka, pragmatyka):
  — rozbiór i generacja, dyskurs, akty mowy i planowanie;
- podejście lingwistyczne w systemach z dostępem w języku naturalnym:
  — tłumaczeniu maszynowym, rozpoznawaniu i generowaniu mowy, generowaniu tekstu, systemach wyszukiwania informacji, budowie słowników, inteligentnych edytorach tekstu;
- reprezentację wiedzy i wnioskowanie:
  — rozumienie języka, automatyczne tworzenie baz wiedzy na podstawie tekstu;
- wyposażenie programowe i sprzętowe do przetwarzania danych językowych,
- narzędzia lingwistyki obliczeniowej do nauki i nauczania języka

Cztery egzemplarze skróconej wersji artykułu (o objętości nie przekraczającej siedmiu stron) wraz z krótkim (pięcioliniowym) streszczeniem należy nadesłać do 10 grudnia 1987 roku na ręce przewodniczącej komitetu programowego konferencji:

Dr Eva Hajicová, Charles University, Faculty of Mathematics, Linguistics

Malostranské n. 25, 118 00 Praha 1, Czechoslovakia.

Autorzy zostaną powiadomieni o przyjęciu artykułów do 28 lutego 1988 roku. Pełne wersje artykułów w postaci gotowej do druku należy dostarczyć organizatorom do 30 kwietnia 1988 r.

Wszelkie zapytania odnośnie programu konferencji, organizacji wystawy i sesji plakatowych można kierować pod adresem:

COLONG 88 Secretariat, c/o MTESZ CONGRESS BUREAU
Kossuth tér 6-8, H-1055 Budapest, Hungary, teleks 225792 MTESZ h

Bezpośrednio po konferencji odbędzie się w Budapeszcie trzeci Kongres EURALEX poświęcony zagadnieniom leksykografii. (MM)

TADEUSZ MORZY
Instytut Automatyki
Politechnika Poznańska

Synchronizacja transakcji w systemach rozproszonych baz danych (I)

Istotna nowość w problematyce systemów rozproszonych baz danych (SRBD), w porównaniu z systemami scentralizowanych baz danych, leży w sferze zagadnień dotyczących zarządzania tymi systemami. Jest ono realizowane przez system zarządzania rozproszoną bazą danych (SZRBD), którego celem jest między innymi przesłonienie użytkownikom rozproszonej struktury bazy danych oraz wywołanie wyrażenia, że korzystają oni z bazy monolitycznej i scentralizowanej. Dzięki maskowaniu przez SZRBD faktu rozproszenia bazy danych, problemy występujące na poziomie oprogramowania użytkowego w SRBD nie różnią się zasadniczo od analogicznych problemów w scentralizowanych bazach danych. Również szeroko rozumiane problemy modelu pojęciowego bazy danych nie ulegają zasadniczym zmianom. Natomiast problemy zarządzania rozproszoną bazą danych mają istotną specyfikę i są jakościowo nowe.

Jak wynika z samej istoty SRBD, systemy te stanowią środowisko wielokomputerowe, wieloprogramowe i wielodostępne. Stąd jednym z najistotniejszych i jednocześnie najtrudniejszych problemów stojących przed projektantem SZRBD jest problem poprawnego zarządzania współbieżnym dostępem wielu użytkowników do rozproszonej bazy danych. Problem ten nosi w literaturze nazwę **problemu synchronizacji**. Elementarną jednostką interakcji użytkownika z systemem bazy danych jest tzw. **transakcja** [1], mówi się zatem o problemie **synchronizacji transakcji**. Ogólnym celem algorytmu synchronizacji transakcji jest zapewnienie spójności rozproszonej bazy danych oraz zagwarantowanie zrealizowania każdej transakcji zainicjowanej w SRBD. Oczywiste jest przy tym dążenie do konstrukcji algorytmów synchronizacji o maksymalnej efektywności i to zarówno w sensie ich minimalnej złożoności obliczeniowej, jak i w sensie maksymalnej przepustowości systemu. Przypomnijmy, że problem zapewnienia spójności bazy danych polega na zapewnieniu poprawności danych, zawartych w bazie danych, oraz poprawności wzajemnych związków pomiędzy tymi danymi. Inaczej mówiąc, problem zapewnienia spójności bazy danych polega na ochronie danych przed możliwością wystąpienia błędu, w wyniku ich niewłaściwego uaktualnienia, wprowadzania lub usunięcia. Spójność bazy danych może być naruszona zasadniczo z trzech powodów.

(i) na skutek błędnego zarządzania współbieżnym dostępem wielu użytkowników do tych samych danych,

(ii) na skutek wystąpienia awarii SBD powodującej częściowy lub całkowity upadek systemu,

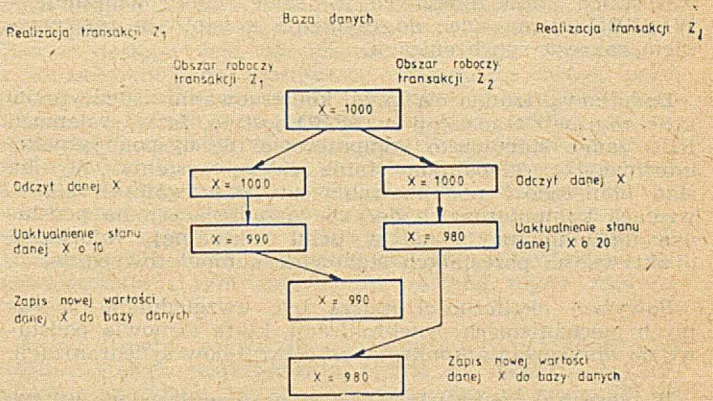
(iii) na skutek błędnego programu użytkownika.

Problemy odporności systemu bazy danych na wystąpienia awarii sprzętowych lub błędów programowych stanowią odrębne, ważne zagadnienie wykraczające jednakże znacznie poza ramy tego artykułu. W dalszym ciągu będzie rozpatrywana możliwość naruszenia spójności bazy danych, w wyniku błędnego zarządzania współbieżnym dostępem zadań użytkowych do tych samych danych.

Poniżej przedstawiono dwa klasyczne przykłady naruszenia spójności bazy danych w wyniku nie kontrolowanej współbieżnej realizacji transakcji [3].

Przykład 1. Zagubione uaktualnienie

Rozważmy przykładowy system inwentaryzacji magazynowej. Załóżmy, że w systemie tych dwóch użytkowników realizuje jednoczesny dostęp do pewnej danej **X** oznaczającej liczbę aktualnie dostępnych elementów (np. przekaźników) w magazynie. Załóżmy, że pierwszy z użytkowników pobiera z magazynu 10 sztuk przekaźników (zadanie Z_1), natomiast drugi pobiera 20 sztuk (zadanie Z_2). Współbieżna realizacja transakcji Z_1 i Z_2 przedstawiona na rys. 1 jest niepoprawna, gdyż zagubiona została informacja o pobraniu z magazynu 10 sztuk przekaźników przez pierwszego z użytkowników. Poprawny stan danej **X** powinien wynosić 1000 — 10 — 20 = 970.

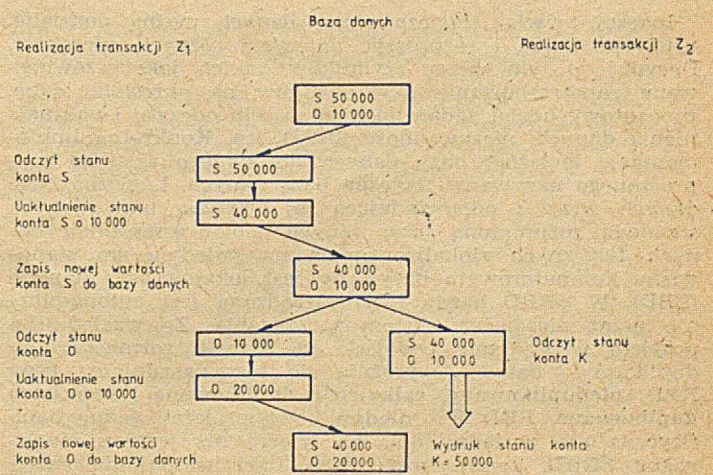


Rys. 1. Przykład wystąpienia anomalii „zagubionego uaktualnienia"

Przykład 2. Błędny odczyt

Rozważmy hipotetyczny system bankowy, w którym każde konto **K** klienta składa się z dwóch części: subkonta stałego **(S)** oraz subkonta obrotowego **(O)**, **K = S + O**. Załóżmy, że w systemie realizowane są następujące dwie transakcje żądające dostępu do tego samego konta **K**:

— transakcja Z_1 realizująca przelew kwoty 10 000 zł z subkonta stałego **S** konta **K** ma subkonto **O** tego samego konta **K**;



Rys. 2. Przykład wystąpienia anomalii „błędnego odczytu"

— transakcja Z_2 realizująca odczyt łącznego stanu konta **K**, tj. sumy kont **S** i **O**.

Jak łatwo zauważyć współbieżna realizacja transakcji Z_1 i Z_2 przedstawiona na rys. 2 jest niepoprawna, gdyż prowadzi do błędnego wydruku stanu konta **K**. Realizowany przez transakcję Z_1 przelew w obrębie konta **K** nie zmienia sumarycznego stanu tego konta. Zatem transakcja Z_2 powinna wydrukować stan 60 000 zł, a nie 50 000 zł.

Przedstawione przykłady ilustrują możliwość naruszenia tzw. spójności wewnętrznej scentralizowanej lub lokalnej bazy danych. W wypadku rozproszonej bazy danych (RBD) zapewnienie jej spójności wymaga dodatkowo:

— zapewnienia spójności wewnętrznej danych przechowywanych w różnych lokalnych bazach danych (np. ze względów administracyjnych informacje o koncie **S** i koncie **O** mogą być rozdzielone pomiędzy dwie różne lokalne bazy danych),

— spójności zewnętrznej, rozumianej jako identyczność wszystkich kopii fizycznych tej samej danej logicznej.

W SRBD mamy więc do czynienia z istotnym rozszerzeniem zakresu synchronizacji.

Dodatkową trudnością przy konstruowaniu algorytmów synchronizacji transakcji w SRBD jest to, że w systemach RBD żadne stanowisko komputerowe nie dysponuje pełną informacją o globalnym stanie całego systemu. Wynika stąd konieczność podejmowania na poszczególnych stanowiskach komputerowych decyzji zarządzających na podstawie niekompletnej i nie w pełni aktualnej informacji o aktywności pozostałych stanowisk komputerowych.

Powyższe okoliczności muszą być uwzględniane w samych mechanizmach synchronizacji, które stanowią podstawę do opracowania konkretnych algorytmów synchronizacji.

W pierwszej części artykułu zostaną przedstawione wstępne zagadnienia dotyczące metod synchronizacji transakcji w SRBD. Przegląd tych metod oraz charakterystyka nowych tendencji w zakresie synchronizacji transakcji w SRBD będą przedmiotem drugiej części artykułu.

PODSTAWOWE POJĘCIA I DEFINICJE

Najogólniej, system rozproszonej bazy danych definiuje się jako trójkę (RBD, τ, C(τ)), gdzie RBD oznacza rozproszoną bazę danych, τ jest zbiorem aktualnie realizowanych transakcji, a C(τ) stanowi kryterium poprawności realizacji zbioru transakcji τ w RBD. Rozproszona baza danych stanowi zbiór danych, do których użytkownicy żądają dostępu, przy czym zbiór ten jest fizycznie rozproszony na różnych, wzajemnie oddalonych stanowiskach komputerowych SRBD. Formalnie RBD można rozpatrywać z dwóch punktów widzenia: logicznego i fizycznego. Z logicznego punktu widzenia RBD stanowi zbiór tzw. danych logicznych (nazywany logiczną bazą danych), za pomocą których użytkownik SRBD definiuje swoje zadanie zwane transakcją. Użytkownik pozostaje przy tym nieświadomy faktu fizycznego rozproszenia i ewentualnej duplikacji danych, do których żąda dostępu. Logiczna baza danych może być fizycznie zaimplementowana na wiele sposobów, w zależności od przyjętej konfiguracji SRBD.

Inaczej mówiąc, logiczną bazę danych można podzielić, utworzyć kopię i rozproszyć na wiele różnych sposobów. Decyduje o tym szereg czynników, takich jak: przewidywane zapotrzebowanie użytkowników na określone dane, procentowy udział żądań dostępu w celu odczytu i uaktualnienia danych, niezawodność SRBD, itp. Konkretną implementację logicznej bazy danych dla danego systemu rozproszonego nazywamy fizyczną bazą danych. Logiczna baza danych wraz z odpowiadającą jej fizyczną bazą danych stanowią rozproszoną bazę danych. Zbiór wszystkich danych fizycznych zlokalizowanych na pojedynczym stanowisku komputerowym nazywany jest lokalną bazą danych (LBD). W SRBD każda LBD zarządzana jest autonomicznie przez niezależny system zarządzania. Ze względu na przyjęte w systemie podejście do kwestii rozmieszczania duplikatów danych, wyróżnia się trzy zasadnicze typy RBD: **nieduplikowane, całkowicie duplikowane** i **częściowo duplikowane.** RBD jest nieduplikowana, jeżeli żadna dana fizyczna nie posiada kopii na więcej niż jednym stanowisku SRBD. Jeżeli każda dana fizyczna posiada swoją kopię na wszystkich stanowiskach SRBD, to taka RBD jest całkowicie duplikowaną RBD. Jeżeli istnieją dane fizyczne, które mają kopie zlokalizowane na dwóch lub więcej stanowiskach systemu komputerowego, to taka RBD jest częściowo duplikowana.

Należy zwrócić uwagę na fakt, że pojęcie rozproszenia odnosi się w równej mierze do modelu danych, jak i systemu zarządzania rozproszoną bazą danych. Zarządzanie realizowane jest bowiem w sposób zdecentralizowany i rozproszony przez SZRBD, który organizuje współpracę równorzędnych i autonomicznych systemów zarządzania LBD. Fakt istnienia autonomicznych systemów zarządzania lokalnymi bazami danych stanowi jedną z podstawowych cech odróżniających systemy RBD od systemów baz scentralizowanych, w szczególności od zrealizowanych w systemach wielokomputerowych.

Jak już wspomniano elementarną jednostką interakcji użytkownika z SZRBD jest **transakcja.** Najogólniej transakcję definiuje się jako ciąg instrukcji zawierający operacje dostępu do bazy danych, ograniczony znacznikiem początku i końca transakcji. Transakcję powinny cechować następujące własności: spójność, niepodzielność, trwałość i niezależność od efektu domina [1]. Koncepcja transakcji pozwala rozłącznie rozpatrywać problemy synchronizacji, poprawności dostępu do danych oraz niezawodności. Warto zauważyć, że przedstawiona wyżej nieformalna definicja transakcji odnosi się do poziomu fizycznego RBD, tj. do fizycznej bazy danych. Reprezentowany jest w niej zatem punkt widzenia SZRBD. Jednakże, jak już wcześniej wspomniano, transakcja użytkownika jest zdefiniowana w odniesieniu do poziomu logicznego RBD. Dlatego konieczne jest rozróżnianie pomiędzy **transakcją fizyczną** (krótko — transakcją) a **transakcją logiczną** reprezentującą punkt widzenia użytkownika SRBD. Odwzorowanie transakcji logicznej w transakcję fizyczną jest niejednoznaczne w tym sensie, że pojedynczej transakcji logicznej odpowiada pewien zbiór transakcji fizycznych różniących się: lokalizacją realizacji elementarnych operacji, kierunkiem przesyłania danych, itp. Wybór określonej transakcji fizycznej (dla danej transakcji logicznej) jest wykonywany przez procedury optymalizacyjne SZRBD.

WSPÓŁBIEŻNA REALIZACJA TRANSAKCJI

W przykładach 1 i 2 pokazano, że niekontrolowane współbieżne wykonanie zbioru transakcji może prowadzić do naruszenia spójności bazy danych. Z tego powodu współbieżne wykonywanie transakcji powinno być odpowiednio synchronizowane. Formalny opis wykonania zbioru transakcji w bazie danych jest nazywany **realizacją.** Dowolna realizacja jest wynikiem działania określonego algorytmu synchronizacji transakcji przyjętego w SRBD.

Podstawową miarą oceny jakości działania danego algorytmu synchronizacji jest rozmiar zbioru realizacji poprawnych generowanych przez ten algorytm. Jednakże zbiór realizacji poprawnych dla danego algorytmu synchronizacji jest funkcją informacji o SRBD, jaką ten algorytm synchronizacji rozporządza. Informacje te mogą dotyczyć struktury logicznej i fizycznej RBD, ograniczeń integralnościowych nałożonych na RBD, charakteru obliczeń wykonywanych przez transakcje itp. Zakres i charakter tych informacji jest określony przez przyjęty model współbieżności. Model ten precyzuje kryterium poprawności C(τ).

Klasycznym i najczęściej spotykanym modelem współbieżności jest tzw. **jednowersyjny model syntaktyczny.** Zakłada się w nim, że algorytm synchronizacji dysponuje wyłącznie informacją czysto syntaktyczną o zbiorze realizowanych transakcji, tj. znane mu są jedynie zbiory danych, do których transakcje żądają dostępu. Ogólnie przyjętym kryterium poprawności współbieżnej realizacji zbioru transakcji w tym modelu jest **kryterium uszeregowalności.** Zgodnie z tym kryterium dowolna współbieżna realizacja zbioru transakcji jest poprawna, jeżeli jest ona równoważna dowolnej realizacji sekwencyjnej tego zbioru. Przez realizację sekwencyjną rozumiana jest taka realizacja zbioru transakcji, która spełnia warunek, że w każdej chwili w systemie jest wykonywana co najwyżej jedna transakcja.

Problem uszeregowalności dowolnej realizacji współbieżnej, tj. problem określenia czy dla danej realizacji współbieżnej istnieje równoważna jej realizacja sekwencyjna, jest problemem obliczeniowo trudnym (należy do klasy problemów NP-zupełnych). Dlatego kryterium uszeregowalności zastępuje się w praktyce kryterium mocniejszym,

ale testowalnym w czasie wielomianowym. Kryterium to nosi nazwę **D-uszeregowalności** i stanowi warunek dostateczny uszeregowalności [1]. Zgodnie z tym kryterium, dla dowolnych dwóch transakcji T_i i T_j żądających dostępu do tych samych danych, jeżeli wykonanie operacji $O_i \in T_i$ poprzedza na k-tym stanowisku systemu wykonanie operacji $O_j \in T_j$, to w celu zagwarantowania poprawności realizacji obu transakcji, wykonanie dowolnej operacji transakcji T_i na dowolnym stanowisku (łącznie z k-tym) musi poprzedzać wykonanie dowolnej operacji transakcji T_j. Inaczej mówiąc, wykonanie operacji transakcji T_i i T_j musi odbywać się na wszystkich stanowiskach systemu w ustalonym porządku. Jak już wspomniano, zdefiniowane wyżej kryteria uszeregowalności i D-uszeregowalności zostały sformułowane przy wykorzystaniu informacji wyłącznie czysto syntaktycznej. W wypadku, gdy jest dostępna pewna dodatkowa informacja, np. o strukturze danych, o ograniczeniach integralnościowych czy o zbiorze przetwarzanych transakcji, kryterium uszeregowalności można osłabić uzyskując w wyniku większy stopień współbieżności wykonywania transakcji.

KONFLIKTY DZIAŁANIA SRBD

Kryterium uszeregowalności nie jest wystarczającym kryterium poprawności działania systemów rozproszonej bazy danych, a tym samym poprawności synchronizacji transakcji w SRBD. Jak wspomniano we wstępie, problem poprawności działania SRBD należy rozpatrywać w ujęciu globalnym jako:

(i) problem zapewnienia spójności i integralności RBD,

(ii) problem zapewnienia zrealizowania każdej transakcji zainicjowanej w systemie.

Przyjęcie określonej metody synchronizacji transakcji w SRBD zapewniającej spójność wewnętrzną i zewnętrzną RBD (warunek (i)) może pociągnąć za sobą możliwość wystąpienia w systemie tzw. konfliktów działania SRBD (ang. system performance failures). Wyróżnia się cztery typy tych konfliktów, a mianowicie: **martwy punkt, cykliczny restart, stałe zablokowanie** i **stałe restartowanie.** Wystąpienie w systemie któregokolwiek z powyższych konfliktów działania prowadzi w konsekwencji do niespełnienia warunku (ii), tj. do niezakończenia wykonywania transakcji lub zbioru transakcji zainicjowanych w systemie. Niezrealizowanie transakcji prowadzi w pewnym sensie również do naruszenia spójności RBD, rozumianej szeroko jako relacja odpowiedzialności pomiędzy „światem zewnętrznym" a jego abstrakcyjnym odzwierciedleniem, jakim jest RBD.

Konflikt martwego punktu jest definiowany jako taki stan systemu, w którym dwie lub więcej transakcji oczekuje wzajemnie na uwolnienie danych niezbędnych do zakończenia ich wykonywania. Warto zwrócić uwagę na fakt, że w SRBD stan martwego punktu może wystąpić lokalnie (na pojedynczym stanowisku komputerowym) lub globalnie (w obrębie całego SRBD), obejmując jednocześnie wiele stanowisk komputerowych. W konsekwencji w wypadku SRBD rozwiązanie tego problemu jest znacznie trudniejsze niż w wypadku systemów scentralizowanych baz danych. Wyróżnia się dwie ogólne metody rozwiązania tego problemu w SRBD, mianowicie metodę wykrywania i likwidacji oraz metodę unikania.

Metoda wykrywania i likwidacji polega na okresowym sprawdzaniu czy SRBD aktualnie znajduje się w stanie martwego punktu. W tym celu konstruowany jest graf czekania transakcji i badane jest istnienie cyklu w tym grafie. W wypadku implementacji tej metody w SRBD zasadniczym problemem jest efektywne konstruowanie grafu czekania transakcji.

Metoda unikania martwego punktu polega na sprawdzaniu, czy dwie transakcje będące wzajemnie w konflikcie dostępu do danych mogą znajdować się w relacji czekania, czy też jedna z tych transakcji musi zostać wycofana, aby nie spowodować wystąpienia stanu martwego punktu. Najprostszym przykładem odpowiedniej procedury testującej jest procedura, która każdorazowo w momencie zgłoszenia niekompatybilnego żądania dostępu do danej powoduje natychmiastowe wycofanie transakcji zgłaszającej takie żądanie.

Konflikt stałego zablokowania zachodzi wówczas, gdy na skutek inicjowania w systemie w nieznanych a priori momentach czasu coraz nowych transakcji z nieskończonego ich strumienia, transakcja lub zbiór transakcji nigdy nie uzyskają dostępu do wszystkich danych, niezbędnych do ich wykonania i tym samym nie zostaną nigdy zrealizowane. Stosowane w SRBD metody ochrony przed wystąpieniem konfliktu stałego zablokowania nie różnią się w istocie od znanych metod stosowanych w systemach scentralizowanych. Należy do nich metoda zapobiegania, polegająca na wymuszeniu kolejności dostępu do danych zgodnie z kolejnością inicjowania transakcji w SRBD, oraz metoda wykrywania i likwidacji, polegająca na pomiarze pewnych parametrów transakcji przebywających w systemie i określeniu na tej podstawie, które transakcje są w stanie stałego zablokowania. Likwidacja stanu stałego zablokowania transakcji polega wówczas najczęściej na czasowym przerwaniu inicjowania nowych transakcji.

Powyższe dwa rodzaje konfliktów działania mają tę właściwość, że mogą powstać w wypadku wystąpienia konfliktu dostępu do danych pomiędzy dwoma transakcjami oraz, że jedna z tych transakcji czeka na zakończenie realizacji drugiej. Specyfika systemu rozproszonego, charakteryzującego się rozproszeniem fizycznych zasobów systemowych oraz autonomią stanowisk komputerowych, powoduje, że podstawowym mechanizmem rozwiązania takich konfliktów dostępu jest odrzucanie transakcji. Prowadzi to w konsekwencji do możliwości wystąpienia dwóch pozostałych konfliktów działania, a mianowicie konfliktów cyklicznego i stałego restartowania. **Konflikt cyklicznego restartowania** zachodzi wówczas, gdy dwie lub więcej transakcji powodują wzajemne wycofanie lub wywłaszczenie. **Konflikt stałego restartowania** zachodzi wówczas, gdy na skutek nieskończonego strumienia nowych transakcji inicjowanych w systemie w nieznanych a priori momentach, transakcja lub zbiór transakcji są stale wycofywane lub wywłaszczane i tym samym nie zostaną nigdy zrealizowane.

Jedna z metod rozwiązywania konfliktów cyklicznego restartu i stałego restartowania, oparta na metodzie wykrywania i likwidacji została zaproponowana w pracy [2]. W tej metodzie każda transakcja T_j otrzymuje jednoznaczny identyfikator oznaczony przez TS(T), związany z czasem jej inicjowania. W systemie określa się dopuszczalną liczbę restartów dla transakcji. Po przekroczeniu tej liczby restartów dana transakcja T_j jest uważna za znajdującą się w stanie stałego lub cyklicznego restartowania. W takiej sytuacji następuje zaetykietowanie tej transakcji oraz wszystkich danych, do których żąda ona dostępu. Etykietowanie danej x polega na przydzieleniu jej znacznika czasowego (oznaczanego mark (x)), równego identyfikatorowi TS(Tj) zaetykietowanej transakcji.

Warunek wstępny dostępności danej x dla dowolnej transakcji T_i można sformułować w postaci:

$$TS(T_i) \leqslant \text{mark}(x)$$

Przyjmuje się, że znacznik czasowy każdej niezaetykietowanej danej x jest równy $+\infty$. Oczywiście wówczas zachodzi związek:

$$\bigwedge_i TS(T_i) < \text{mark}(x)$$

Etykietowanie danych jest realizowane zgodnie z następującym schematem.

if $\text{mark}(x) > TS(T_i^*)$

then $\text{mark}(x) := TS(T_i^*)$,

gdzie T_i^* oznacza zaetykietowaną transakcję T_i, a $TS(T_i^*)$ — etykietę czasową tej zaetykietowanej transakcji.

Włączenie powyższej metody rozwiązania konfliktów cyklicznego i stałego restartowania do metody blokowania nie nastręcza żadnych trudności. Mianowicie transakcja T_i może założyć blokadę danej x, jeżeli spełnione są dwa warunki:

(i) $TS(T_i) \leqslant \text{mark}(x)$,

(ii) typ blokady żądanej przez T_i jest kompatybilny z aktualną blokadą danej x.

Warunek (i) wprowadza dodatkowe ograniczenie dostępności danej x. Uniemożliwia on dostęp do tej danej strumieniowi nowo zainicjowanych transakcji. Wynikające stąd wstrzymanie realizacji nowych transakcji nie jest jednak całkowite — transakcje żądające dostępu do innych danych mogą być normalnie wykonywane.

Jak wynika z przeprowadzonych badań symulacyjnych opisana metoda może przyczynić się do zwiększenia efektywności działania systemu nawet wówczas, gdy nie za-

chodzi wypadek stałego restartowania transakcji. Ogranicza ona bowiem liczbę restartów transakcji, wskutek czego zmniejsza się średni czas odpowiedzi, będący jednym z najważniejszych kryteriów oceny działania SRBD. Skuteczność tej metody zależy od właściwego doboru granicznej liczby restartów, której przekroczenie jest traktowane jako symptom wystąpienia konfliktów stałego lub cyklicznego restartowania.

EWA GURBIEL
HELENA KRUPICKA
ZDZISŁAW PŁOSKI
Wrocław

LITERATURA

[1] Cellary W., Morzy T.: Problemy i metody synchronizacji w systemach rozproszonych baz danych. Archiwum Automatyki i Telemechaniki. T. XXX, nr 5—6, 1986
[2] Cellary W., Morzy T.: Locking with Prevention of Cyclic and Infinite Restarting in Distributed Database Systems. Proc. 11th Int. Conf. VLDB, Stockholm, 1985
[3] Bernstein P. A., Goodman N.: Concurrency Control in Distributed Database Systems. ACM Computing Surveys, Vol. 11, No. 2, pp. 185—221, 1981.

Składnia języka Logo — próba formalizacji (2)

W pierwszej części opisu składni języka Sinclair Logo przedstawiono składnię opisów, instrukcji i składnię procedur pierwotnych. Poniżej omówiono składnię wyrażeń, funkcji pierwotnych i obiektów.

SKŁADNIA WYRAŻEŃ

Mówiąc ogólnie, w Logo wartościami wyrażeń są *słowa* i *listy*. W opisie składni wyróżnia się wyrażenia słowowe i listowe. Istnienie typowych operatorów działań arytmetycznych i logicznych, z właściwą im hierarchią, uprawnia do dalszego różnicowania składni wyrażeń słowowych — wprowadzenia wyrażeń arytmetycznych i logicznych. Odnotujmy, iż na poziomie *nazwy wartości* różnice te nie występują.

wyrażenie:
  wyrażenie-słowowe
  wyrażenie-listowe
wyrażenie-słowowe:
  wyrażenie-arytmetyczne
  wyrażenie-logiczne
  wyrażenie-napisowe
wyrażenie-arytmetyczne:
  składnik
  wyrażenie-arytmetyczne operator-typu-d składnik
operator-typu-d:
  \+
  —
składnik:
  czynnik
  składnik operator-typu-m czynnik
operator-typu-m:
  *
  /
czynnik:
  proste-wyrażenie-arytmetyczne
  wyrażenie arytmetyczne
  — *czynnik*
proste-wyrażenie-arytmetyczne:
  liczba
  "stała-liczbowa
  nazwa-wartości

Po znaku " nie może wystąpić odstęp.

wyrażenie-logiczne:
  proste-wyrażenie-logiczne
  relacja
proste-wyrażenie-logiczne:
  "TRUE *nazwa-wartości*
  "FALSE *wyrażenie-logiczne*

Po znaku " nie może wystąpić odstęp. TRUE i FALSE mogą być pisane dowolną kombinacją małych i wielkich liter.

relacja:
  wyrażenie-arytmetyczne operator-typu-n wyrażenie arytmetyczne
  wyrażenie = wyrażenie-arytmetyczne
  wyrażenie = proste-wyrażenie-logiczne
  wyrażenie = wyrażenie-napisowe
  wyrażenie = wyrażenie-listowe
operator-typu-n:
  <
  =
  >
wyrażenie-napisowe:
  "napis
  "ogranicznik symbol...
  "ogranicznik
  nazwa-wartości
  (wyrażenie-napisowe)

Jeśli ciąg *symbol...* tworzy liczbę, to poprzedzający go ogranicznik musi być różny od znaku —. Po znaku " nie może wystąpić odstęp. *Napis*, *ogranicznik* i *symbol* zdefiniowano w punkcie: składnia obiektów.

wyrażenie-listowe:
  lista
  nazwa-wartości
  wyrażenie-listowe

W definicjach wyrażeń występuje pojęcie *nazwa wartości*.

nazwa-wartości:
  nazwa-wartości-zmiennej
  nazwa-wartości-funkcji
nazwa-wartości-zmiennej:
  :nazwa-zmiennej

Po znaku : nie może wystąpić odstęp.

nazwa-zmiennej:
Nazwą zmiennej może być każde niepuste *słowo* różne od *liczby* i *ogranicznika* (p. punkt: składnia obiektów).

nazwa-wartości-funkcji:
  funkcja-pierwotna
  nazwa-funkcji
  nazwa-funkcji argument...
nazwa-funkcji:
  nazwa-procedury
argument:
  parametr-aktualny

Nazwa wartości występuje w wyrażeniach: arytmetycznym, logicznym, napisowym i listowym. Wartością zmien-

nej bądź funkcji w czasie interpretowania wymienionych wyrażeń musi być odpowiednio: *liczba*, TRUE lub FALSE, *napis*, *lista*. Wymaganie to nie wynika ze składni wyrażeń, lecz powinno być uwzględnione przez użytkownika.

SKŁADNIA FUNKCJI PIERWOTNYCH

Funkcje pierwotne mogą występować tylko w kontekstach wyrażeń bądź tworzyć samoistne wyrażenia. Pogrupowano je dalej według rodzajów reprezentowanych przez nie wyników.

funkcja-pierwotna:
  funkcja-arytmetyczna
  funkcja-logiczna
  funkcja-listowa
  funkcja-słowowa
  funkcja-obiektowa

W grupie funkcji arytmetycznych można wyodrębnić cztery podgrupy: funkcje związane z grafiką Logo, funkcje trygonometryczne i odwrotne do nich, funkcje dublujące i uzupełniające operatory arytmetyczne oraz pozostałe.

funkcja-arytmetyczna:
  XCOR TOWARDS *współrzędne-punktu*
  YCOR BackGround
  HEADING PenColour
  SINe *kąt* ARCSIN *wyrażenie-arytmetyczne*
  COSine *kąt* ARCCOS *wyrażenie-arytmetyczne*
  TANgent *kąt* ARCTANgent *wyrażenie-arytmetyczne*
  COTangent *kąt* ARCCOTangent *wyrażenie-arytmetyczne*
SUM *wyrażenie-arytmetyczne wyrażenie-arytmetyczne*
  (SUM *wyrażenie-arytmetyczne wyrażenie-arytmetyczne...*)
PRODUCT *wyrażenie-arytmetyczne wyrażenie-arytmetyczne*
  (PRODUCT *wyrażenie-arytmetyczne wyrażenie-arytmetyczne...*)
DIV *wyrażenie-arytmetyczne wyrażenie-arytmetyczne*
  (DIV *wyrażenie-arytmetyczne wyrażenie-arytmetyczne...*)
REMAINDER *wyrażenie-arytmetyczne wyrażenie-arytmetyczne*
INT *wyrażenie-arytmetyczne*
ROUND *wyrażenie-arytmetyczne*
SQRT *wyrażenie-arytmetyczne*
RANDOM *wyrażenie-arytmetyczne*
COUNT *wyrażenie*
ASCII *wyrażenie-słowowe*
NODES
.EXAMINE *adres-bajtu*
.SERIALIN

Współrzędne punktu są zadane wyrażeniem listowym. Kąt i adres bajtu są wyrażeniami arytmetycznymi. Funkcja .SERIALIN jest „egzotyczna"; pozostaje w związku z instrukcjami sieciowymi.

W grupie funkcji logicznych wyróżniono: alternatywę, koniunkcję, negację, funkcje stałe TRUE i FALSE oraz tzw. funkcje sprawdzające.

funkcja-logiczna:
  OR *wyrażenie-logiczne wyrażenie-logiczne*
    (OR *wyrażenie-logiczne wyrażenie-logiczne...*)
  AND *wyrażenie-logiczne wyrażenie-logiczne*
    (AND *wyrażenie-logiczne wyrażenie-logiczne...*)
  NOT *wyrażenie-logiczne*
  TRUE
  FALSE
  SHOWNP
  EMPTYP *wyrażenie*
  NUMBERP *wyrażenie*
  WORDP *wyrażenie*
  LISTP *wyrażenie*
  MEMBERP *wyrażenie wyrażenie-listowe*
  EQUALP *wyrażenie wyrażenie*
  NAMEP *wyrażenie-słowowe*
  DEFINEDP *wyrażenie-słowowe*
  PRIMITIVEP *wyrażenie-słowowe*
  KEYP

funkcja-listowa:
  POSition
  SCRUNCH
  TextColour
  LIST *wyrażenie wyrażenie*
    (LIST *wyrażenie wyrażenie...*)
  SEntence *wyrażenie wyrażenie*
    (SEntence *wyrażenie wyrażenie...*)
  FPUT *wyrażenie wyrażenie-listowe*
  LPUT *wyrażenie wyrażenie-listowe*
  CURSOR
  .RESERVED
  TEXT *wyrażenie-słowowe*
  ReadList

funkcja-słowowa:
  WORD *wyrażenie-słowowe wyrażenie-słowowe*
    (WORD *wyrażenie-słowowe wyrażenie-słowowe...*)
  CHAR *kod-znaku*
  ReadChar

funkcja-obiektowa:
  FIRST *wyrażenie*
  ButFirst *wyrażenie*
  LAST *wyrażenie*
  ButLast *wyrażenie*
  ITEM *wyrażenie-arytmetyczne wyrażenie-listowe*
  THING *wyrażenie-słowowe*
  IF *wyrażenie-logiczne wyrażenie-listowe wyrażenie-listowe*
  RUN *wyrażenie-listowe*

Wartością funkcji obiektowej jest *obiekt*, tzn. słowo lub lista. READLIST przyjmuje jako wartość tekst wprowadzony z klawiatury. *Kod znaku* jest *wyrażeniem arytmetycznym*. Funkcja RC przyjmuje jako wartość kod znaku przeczytanego z klawiatury.

SKŁADNIA OBIEKTÓW

obiekt:
  słowo
  lista
lista:
  [ ]
  [*obiekt...*]
słowo:
  napis
  liczba
  TRUE
  FALSE

napis:
  puste
  symbol...
  ogranicznik
puste:
Puste oznacza niewystępowanie żadnego znaku. *Napis* utworzony z ciągu symboli powinien być różny od *liczby*, *słowa* TRUE oraz *słowa* FALSE.
symbol:
  litera
  cyfra
  znak-dodatkowy
  dwuznak
  znak-literalny
liczba:
  liczba-dziesiętna E *liczba-całkowita*
  liczba-dziesiętna E + *cyfra...*
  liczba-dziesiętna
liczba-dziesiętna:
  liczba-całkowita .cyfra...
  —*.cyfra*
  .cyfra...
  liczba-całkowita
liczba-całkowita:
  cyfra...
  —*cyfra...*

Po znakach — i . nie może wystąpić odstęp. Litera E (lub e) oznacza dziesiętną podstawę wykładnika.
stała-liczbowa:

Stałą liczbową jest *liczba*, która nie zawiera wewnątrz znaku + ani —. Wprowadzenie tego pojęcia wynika z osobliwości realizacji interpretera Sinclair Logo.
znak-literalny:
  | *znak*
  | *dwuznak*
dwuznak:
  "*ogranicznik*

Po znaku " nie może wystąpić odstęp. Złożenie |*znak* jest traktowane [1]) jako jeden znak literalny (dosłownie). *Ogra-*

[1]) Ze względu na brak w drukarni znaku ukośnika (ang. backslash) w tekście artykułu zastąpiono go kreską pionową.

nicznik lub *znak specjalny* poprzedzony znakiem | przestaje pełnić swoją rolę znaku rozdzielającego — może być elementem *słowa*.

Ogranicznik poprzedzony w słowie znakiem " traci swoje wyróżnione znaczenie.

znak:
litera
cyfra
znak-dodatkowy
znak-specjalny
ogranicznik

litera:

a	n	A	N
b	o	B	O
c	p	C	P
d	q	D	Q
e	r	E	R
f	s	F	S
g	t	G	T
h	u	H	U
i	v	I	V
j	w	J	W
k	x	K	X
l	y	L	Y
m	z	M	Z

cyfra:

0	5
1	6
2	7
3	8
4	9

znak-dodatkowy:

! ' ? {
" , ¤ |
. ↑ }
$: – ~
% ; £ ©
&

znak-specjalny:

[ ] | odstęp

ogranicznik:

( ) + =
< > – /

Alfabet języka Logo składa się z *liter, cyfr, znaków dodatkowych, ograniczników, znaków specjalnych* oraz znaku ⟨ENTER⟩.

LITERATURA

[1] Ross P.: Logo Programming. Addison-Wesley, Reading (MA), 1983
[2] Sparer E.: Sinclair Logo 1 — Turtle graphic. Sinclair Research, Cambridge, 1984
[3] Sparer E.: Sinclair Logo 2 — Programming Reference Manual. Sinclair Research, Cambridge, 1984.

MAREK PAWŁOWSKI
Instytut Informatyki
Politechnika Warszawska

Mikroprogramowanie (2)

W pierwszej części artykułu przedstawiono ogólny schemat blokowy mikroprogramowanego zespołu funkcjonalnego MZF, sposób projektowania układu wykonawczego MZF, a także budowę mikroprocesora segmentowego Am2901. W poniższym artykule omawia się problemy projektowania układu sterującego MZF.

MIKROPROGRAMOWANE UKŁADY STEROWANIA

Centralnym elementem mikroprogramowanego układu sterowania MUS jest pamięć mikroprogramu PM (rys. 1), w której są zapisane słowa sterujące działaniem MZF w kolejnych krokach przetwarzania. W odczytanym z PM słowie sterującym (mikrorozkazowym) wyróżnia się trzy części:

PS — pole sygnałów sterujących działaniem dekodera mikrooperacji warunkowych DMW;

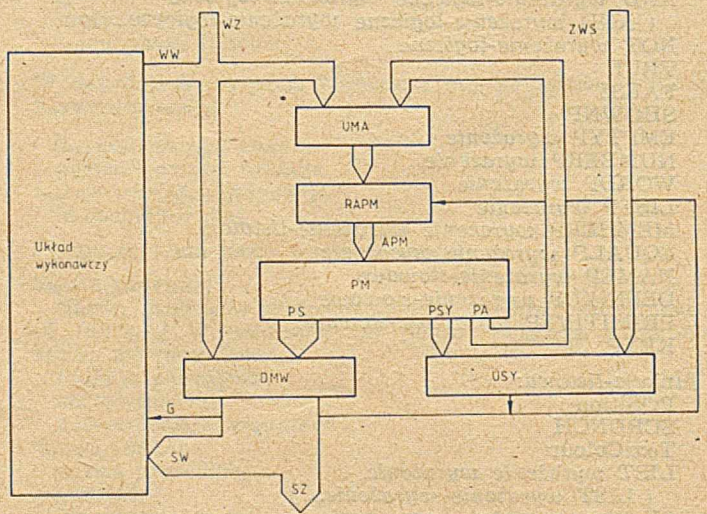
PA — pole sygnałów sterujących obliczaniem adresu następnego mikrorozkazu w układzie modyfikacji adresu UMA;

PSY — pole sygnałów sterujących generowaniem sygnału zegara G w układzie synchronizacji USY.

Dekoder mikrooperacji warunkowych DMW jest konieczny w układzie sterującym, jeżeli w mikroprogramie przewidziano realizację warunkowych mikrooperacji wykonawczych. Przykładowo, warunkowa mikrooperacja wykonawcza może mieć postać:

$A \Leftarrow \{[0]/(Z = 0),\ R/(Z = 1)\}$

Powyższy zapis oznacza wykonanie mikrooperacji $A \Leftarrow [0]$, gdy $Z = 0$ lub $A \Leftarrow R$, gdy $Z = 1$. Sygnał Z jest generowany w układzie wykonawczym w tym samym kroku przetwarzania, w którym ma zostać wykonana jedna z dwóch wymienionych mikrooperacji. Wprowadzenie do mikroprogramu warunkowych mikrooperacji wykonawczych zwykle zmniejsza czas jego wykonania, a także liczbę mikrorozkazów. Sygnały sterujące wykonaniem mikrooperacji $A \Leftarrow [0]$ lub $A \Leftarrow R$ wytwarza blok DMW na podstawie PS i warunku Z. Jeżeli w mikroprogramie nie użyto mikrooperacji warunkowych, to blok DMW jest niepotrzebny



Rys. 1. Ogólny schemat blokowy mikroprogramowanego układu sterowania

i pole PS bezpośrednio steruje układem wykonawczym (sygnały SW) i współpracą z otoczeniem (sygnały SZ).

Zadaniem układu modyfikacji adresu UMA jest obliczenie adresu następnego mikrorozkazu w sposób określony przez pole PA, z uwzględnieniem warunków WW i WZ występujących w mikrooperacji rozejścia aktualnie wykonywanego mikrorozkazu. Budowa bloku UMA jest uzależniona od tego, czy występujące w mikroprogramie warunkowe mikrooperacje rozejścia zawierają etykiety dwóch czy więcej mikrorozkazów. W pierwszym wypadku UMA podaje na wejście rejestru adresowego pamięci mikroprogramu RAPM jeden z dwóch adresów, a w drugim — jeden z wielu adresów odpowiadających położeniu w PM mikrorozkazów występujących w tej samej mikrooperacji rozejścia. W najprostszym wypadku pole PA słowa sterującego może zawierać adresy wszystkich mikrorozkazów bezpośrednio osiąganych z danego mikrorozkazu oraz sterowanie układem warunków określających, który z tych adresów ma zostać zapisany w RAPM. Zaletą takiego rozwiązania jest możliwość dowolnego rozmieszczenia mikrorozkazów w pamięci sterującej PM, wadą natomiast — duża szerokość pola PA wymagająca większej liczby układów scalonych pamięci niż w innych rozwiązaniach.

Gdy z dowolnego mikrorozkazu przechodzi się do jednego z co najwyżej dwóch jego następników, adres jednego z nich można uzależnić od adresu aktualnie wykonywanego mikrorozkazu. Adres drugiego mikrorozkazu może być dowolny i jest wskazywany przez pole PA. Zadaniem bloku UMA jest w tym wypadku zapis do rejestru RAPM:

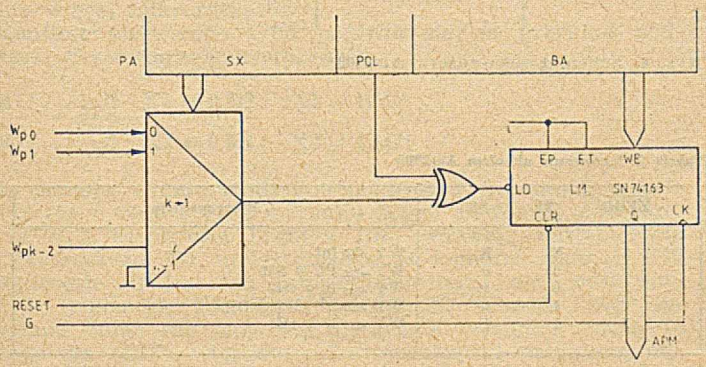
— wartości odczytanej z pola PA (może to być adres skoku w mikroprogramie — BA),

— wartości o 1 większej od aktualnej zawartości rejestru RAPM.

Jeśli połączy się bloki UMA i RAPM, to operacje im przypisane można wykonać w układzie licznika, czyli

$$L := \{BA/(LD = 0),\ INC(L)/(LD = 1)\}$$

przy założeniu, że aktywny jest sygnał zezwalający na zliczanie. Jeżeli na wejście LD licznika poda się warunek decydujący o rozejściu w danym mikrorozkazie, to w zależności od stanu tego warunku na wejściu pamięci mikroprogramu pojawi się adres skoku BA (część pola PA) lub następny adres pamięci. Postać UMA z licznikiem mikrorozkazów LM przedstawiono na rysunku 2.



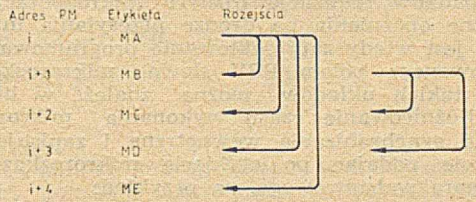
Rys. 2. Układ modyfikacji adresów z licznikiem mikrorozkazów LM, gdzie Wpi jest warunkiem rozejścia, a APM to adres pamięci mikroprogramu

Układ ten zawiera, oprócz wcześniej wymienionych elementów, bramkę EXOR umożliwiającą zmianę polaryzacji warunku. Działanie tego układu można opisać mikrooperacją

$$APM := \{BA/(Wpi \oplus POL = 0) \wedge (RESET = 1),$$
$$INC(APM)/(Wpi \oplus POL = 1) \wedge (RESET = 1),$$
$$[0]/(RESET = 0)\},\quad \text{dla } i = B(SX),$$

gdzie B(SX) oznacza liczbę reprezentowaną w kodzie NB przez słowo SX. W zależności od sygnału POL skok będzie wykonywany w mikroprogramie, gdy Wpi=0 lub Wpi=1 (gdy POL=1). Sygnał RESET umożliwia wyzerowanie licznika, tj. ustawienie w stan początkowy mikroprogramowanego układu sterowania. Aby to wykorzystać, należy pierwszy mikrorozkaz mikroprogramu realizowanego przez MZF umieścić w zerowym słowie pamięci PM.

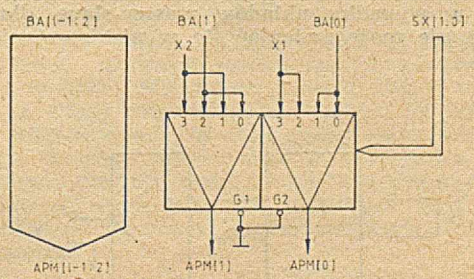
Układ modyfikacji adresu z licznikiem mikrorozkazów jest bardzo prosty i koncepcyjnie najbardziej zbliżony do sposobu adresowania programów w mini- i mikrokomputerach. Dlatego też jest on najczęściej stosowany przy budowie mikroprogramowanych układów sterowania. Nie w każdym jednak MZF jest możliwe jego zastosowanie. Czasem zdarza się, że w mikroprogramie występują mikrorozkazy, z których należy przejść do jednego z wielu jego następników, gdyż warunki decydujące o rozejściu są ważne tylko w danym mikrorozkazie, kolejne rozpatrywanie tych warunków wymagałoby zbyt wielu taktów zegara. Sytuacja taka występuje na przykład w wypadku mikrorozkazu realizującego operację pobrania rozkazu i wskazania początku mikroprogramu odpowiadającego temu rozkazowi.



Rys. 3. Próba umieszczenia następników mikrorozkazu MA w komórkach pamięci bezpośrednio po nim

Podanie w słowie sterującym odczytanym z PM adresów wszystkich następników danego mikrorozkazu jest więc niemożliwe. Powiązanie ich z aktualnym adresem pamięci jest również niemożliwe, co zilustrowano na rysunku 3; nie można zaadresować w przedstawiony sposób następników mikrorozkazu MB, gdyż ich adresy w PM pokryłyby się z adresami następników mikrorozkazu MA. Jedynym więc rozwiązaniem minimalizującym szerokość pola PA jest powiązanie ze sobą adresów następników danego mikrorozkazu.

Adres bazowy występujący w polu PA wskazuje w tej sytuacji początek bloku pamięci programu, w którym są umieszczone mikrorozkazy występujące w danej mikrooperacji rozejścia. Wielkość bloku jest uzależniona od liczby warunków decydujących o rozejściu. Adresy kolejnych mikrorozkazów są określone przez warunki, które są wprowadzone na najmłodsze pozycje słowa adresu pamięci mikroprogramu. Ponieważ w różnych mikrooperacjach rozejścia mogą być brane pod uwagę różne warunki, więc UMA musi zawierać układ warunków sterowany przez SX, zapewniający wprowadzenie odpowiednich warunków na wybrane linie adresowe. Najłatwiej układ ten można wykonać z wykorzystaniem multipleksera (rys. 4, tab. 1).



Rys. 4. Schemat układu modyfikacji adresu ze wstawianiem warunków na pozycje adresowe

Przedstawiony na rysunku 4 blok UMA umożliwia realizację mikrooperacji rozejść bezwarunkowych (SX[1:0]=00), rozejść warunkowych na cztery drogi (SX[1:0]=11) lub na dwie drogi (SX[1:0]=01, 10).

Mikroprogramowany układ sterowania powinien zapewnić wygenerowanie odpowiedniego sygnału synchronizują-

Tabela 1. Działanie układu modyfikacji adresu

SX1	SX0	Mikrooperacje
0	0	APM ⇐ BA
1	1	APM ⇐ BA [l—1:2] ∘ X2 ∘ X1
1	0	APM ⇐ BA [l—1:1] ∘ X1
0	1	APM ⇐ BA [l—1:2] ∘ X2 ∘ BA [0]

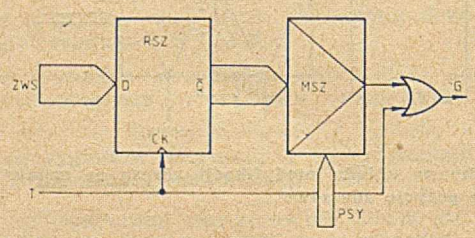
cego działanie wszystkich bloków sekwencyjnych MZF. Okres tego sygnału jest zwykle wyznaczany dla najdłuższej drogi przesyłania informacji w części wykonawczej z uwzględnieniem czasów propagacji sygnałów sterujących w układzie MUS. Może się jednak zdarzyć, że przy tak obliczonym okresie zegara zaprojektowany MZF nie realizuje algorytmu z wystarczającą szybkością. Trzeba wtedy przyspieszyć jego działanie na przykład przez zróżnicowanie czasu trwania poszczególnych mikrorozkazów. W większości wypadków realizacja różnych mikrooperacji wprowadza różne opóźnienia na drodze przesyłania informacji. Konieczne jest wtedy zaprojektowanie programowanego zegara sterowanego polem PSY słowa mikrorozkazowego. Przykłady takich układów można znaleźć w literaturze ([1], [2]). Różnicowanie czasu wykonania mikrorozkazów nazywa się synchronizacją wewnętrzną i zapisuje w mikroprogramie podając po etykiecie mikrorozkazu liczbę taktów zegara podstawowego, na przykład:

MA/3: F ⇐ ADD(X, Y), M ⇐ F, A ⇐ SHL(M, 0), R: = A
exit MB;

O wiele ważniejsze jest rozwiązanie problemu synchronizacji zewnętrznej, umożliwiającej synchronizację pracy wielu mikroprogramowanych zespołów funkcjonalnych. Do synchronizacji zewnętrznej służą sygnały ZWS (rys. 1), powodujące wstrzymanie wykonania mikrorozkazu, w którym są sprawdzane aż do momentu uaktywnienia wybranego sygnału ZWSi. Wstrzymanie to jest realizowane przez zablokowanie zegara G w wysokim stanie. Zasadę działania układu synchronizacji zewnętrznej (rys. 5) omówiono poniżej na przykładzie mikrorozkazu:

MA/Z = 1: A: = DWE exit MB

Zewnętrzny sygnał synchronizujący Z jest na początku równy 0. Układ sterowania wybiera do wykonania mikrorozkaz MA. Na wyjściu pamięci PM pojawia się w polu PSY sterowanie wybierające w multiplekserze MSZ zanegowany sygnał Z, stąd na wejściu bramki sumy wystąpi stan 1. Oznacza to zablokowanie transmisji sygnału taktującego na wyjście bramki sumy i G=1 tak długo, aż pod wpływem narastającego zbocza sygnału T nie zostanie wpisany do rejestru RSZ stan 1 sygnału Z. Nastąpi wtedy odblokowanie sygnału G=T i pojawi się najpierw opadające, a następnie narastające (aktywne) zbocze sygnału G powodujące zapamiętanie w rejestrze A danej wejściowej DWE (synchronizowanej sygnałem Z) i przejście do mikrorozkazu MB. Zablokowanie sygnału zegara w stanie 1 ma znaczenie w wypadku układów master-slave, dla których sterowanie nie może zmieniać się w niskim stanie zegara.



Rys. 5. Schemat blokowy układu synchronizacji zewnętrznej

Funkcje synchronizacji wewnętrznej i zewnętrznej realizuje blok USY zaznaczony na ogólnym schemacie blokowym mikroprogramowanego układu sterowania (rys. 1).

Sekwenser Am2909

Układy modyfikacji adresu są nazywane często sekwenserami. Do realizacji skomplikowanych algorytmów przetwarzania informacji nie zawsze wystarczają skromne możliwości układu z licznikiem mikrorozkazów. W niektórych wypadkach, w celu zmniejszenia pojemności pamięci mikroprogramu, konieczne jest użycie podmikroprogramów. Możliwość taką zapewnia układ Am2909. Jest on 4-bitowym segmentem sekwensera zbudowanego z następujących bloków (rys. 6):

MX — multiplekser źródła adresu,

RD — rejestr adresu (np. adresu początku pętli),

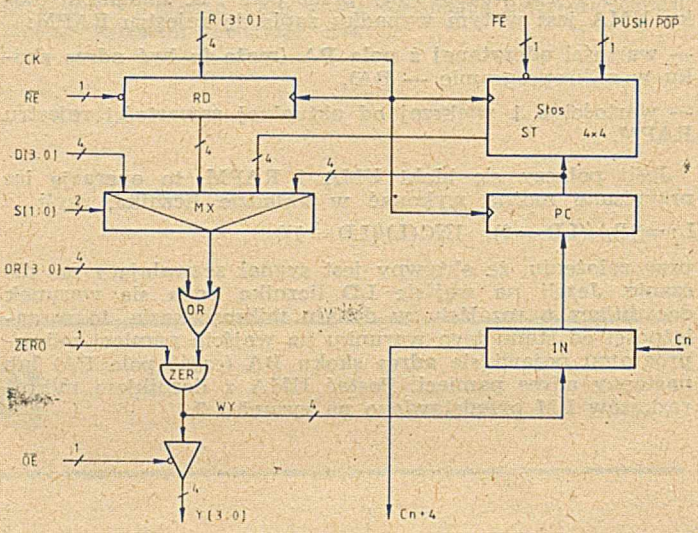
PC — rejestr licznika mikrorozkazów,

IN — układ zwiększenia o 1 adresu pojawiającego się na wyjściu sekwensera,

ST — czterosłowowy stos adresów powrotu z podmikroprogramu,

OR — blok modyfikacji adresu przez wstawianie warunków na linie adresowe,

ZER — układ wymuszania zerowego adresu pamięci mikroprogramu.



Rys. 6. Schemat sekwensera Am2909

Tabela 2. Sterowanie układem Am2909

$\overline{ZERO}$	S1	S0	Mikrooperacje
0	X	X	WY ⇐ [0]
1	0	0	WY ⇐ PC ∨ OR
1	0	1	WY ⇐ R ∨ OR
1	1	0	WY ⇐ ST [SP] ∨ OR
1	1	1	WY ⇐ D ∨ OR

$\overline{FE}$	PUSH/$\overline{POP}$	Mikrooperacje
1	X	SP : = SP
0	1	ST [SP+1] : = PC, SP : = INC (SP)
0	0	SP : = DEC (SP)

Sekwenser Am2909 jest sterowany sygnałami S[1:0], $\overline{FE}$, PUSH/$\overline{POP}$, OR[3:0], $\overline{ZERO}$ w sposób opisany w tabeli 2. Niezależnie od wymienionych wyżej sygnałów sterujących, w układzie Am2909 są wykonywane następujące mikrooperacje:

PC: = AD ([0], WY, Cn), Y ⇐ {WY/($\overline{OE}$ = 0), [Z]/($\overline{OE}$ = 1)},

RD: = R/(RE = 0).

Rejestr licznika mikrorozkazów zawiera zawsze zwiększony o 1 (gdy Cn=1) adres pojawiający się na wyjściu Y sekwensera (gdy $\overline{OE} = 0$). Przy realizacji skoku do podmikroprogramu zawartość PC zostaje zapamiętana w komórce stosu określonej przez zwiększony o 1 wskaźnik stosu ($\overline{FE} = 0$, PUSH/$\overline{POP} = 1$). Adres podmikroprogramu może być wskazany daną z wejścia bezpośredniego D sekwensera Am2909.

Wskaźnik stosu SP wskazuje zawsze ostatnio zapisaną komórkę stosu. Bez względu na operację, na wyjściu stosu jest dostępna informacja zapisana na szczycie stosu (wskazanym przez SP). Stos jest cykliczny czterosłowowy.

Układ Am2909 w typowym zastosowaniu wymaga rejestru na wyjściu pamięci mikroprogramu. Działanie sekwensera może być sterowane bezpośrednio słowem mikrorozkazowym lub za pośrednictwem układu Am29811. Układ Am29811 jest układem kombinacyjnym wytwarzającym sterowanie dla układów Am2909 i licznika pętli, który może być umieszczony w MUS. Czterobitowe słowo sterujące doprowadzone do Am29811 umożliwia wybór jednej z szesnastu możliwych realizacji mikrooperacji rozejścia w mikroprogramie. Wśród tych możliwości dostępne są również operacje warunkowe pozwalające na prostą realizację pętli w mikroprogramie. Układy Am2909 i Am29811 zostały dokładnie omówione w [2] i [6].

OKREŚLANIE ZAWARTOŚCI PAMIĘCI MIKROPROGRAMU

Jednym z ostatnich etapów projektowania układu mikroprogramowanego jest wyznaczenie zawartości pamięci sterującej. Czynność ta wymaga:

- przyporządkowania mikrorozkazom danego mikroprogramu adresów pamięci,
- wypełnienia pamięci słowami dwójkowymi odpowiadającymi kolejnym mikrorozkazom mikroprogramu.

Podczas rozmieszczania mikrorozkazów w pamięci PM mogą wystąpić sytuacje uniemożliwiające umieszczenie w PM danego mikroprogramu. Do sytuacji tych, nazywanych konfliktami rozmieszczania, zalicza się:

- wystąpienie grupy mikrorozkazów wymagających nieskończonej liczby kolejnych adresów pamięci,
- próbę umieszczenia w tym samym słowie pamięci dwóch lub więcej mikrorozkazów.

Konflikt pierwszego typu pojawia się na przykład podczas adresowania mikrorozkazów:

M1: ... exit X1 = 0.M2, X1 = 1.M1;

M2: ... exit X2 = 0.M1, X2 = 1.M2;

za pomocą układu modyfikacji adresu z licznikiem mikrorozkazów. Gdy sygnał warunku X1=1 powoduje zwiększenie o 1 zawartości licznika LM, wówczas następnikiem mikrorozkazu M1 umieszczonego w komórce pamięci PM o adresie (i) jest ten sam mikrorozkaz, ale odczytany z (i+1) komórki PM. Ponieważ stan X1=1 może trwać dowolnie długo (np. gdy jest to sygnał warunku zewnętrznego dla MZF), więc dla mikrorozkazu M1 należałoby zarezerwować bardzo dużą liczbę adresów, przekraczającą rozsądną wielkość pamięci mikroprogramu. Aby tego uniknąć, trzeba dla X1=1 wykonać zapis do LM adresu mikrorozkazu M1. Wtedy dla X1=0 nastąpi wykonanie operacji LM:=INC(LM) i mikrorozkaz M2 należy umieścić w pamięci PM pod adresem (i+1). Podobnie można postąpić przy adresowaniu mikrorozkazów występujących w mikrooperacji rozejścia M2; otrzymuje się wówczas następującą sekwencję mikrorozkazów (rys. 7).

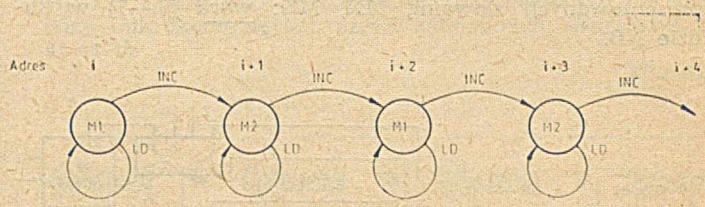
Sekwencja ta wymaga nieskończonej liczby kolejnych adresów pamięci PM; aby rozwiązać ten problem należy zmodyfikować mikroprogram dopisując do niego mikrorozkaz z bezwarunkową mikrooperacją rozejścia, zastępujący mikrorozkaz M1 w mikrooperacji rozejścia M2:

M1: ... exit X1 = 0.M2; X1 = 1.M1;

M2: ... exit X2 = 0.MD; X2 = 1.M2;

MD: exit M1;

Przejście z MD do M1 musi nastąpić przez zapis do licznika LM adresu mikrorozkazu M1, a część wykonawcza mikrorozkazu MD powinna zapewnić wykonanie mikrooperacji tożsamościowych („nic nie rób") we wszystkich blokach układu wykonawczego.



Rys. 7. Przykładowa sekwencja mikrorozkazów w pamięci sterującej

Drugiego typu konflikt rozmieszczania mikrorozkazów występuje na przykład w wypadku mikroprogramu:

MA: ... exit X1 = 0. MB, X1 = 1.MC;

MB: ... exit X1 = 0, MA, X1 = 1.MC;

MC : ...

Założono, że pamięć mikroprogramu jest adresowana przez układ UMA, pokazany na rysunku 4. Warunki wykonania mikrooperacji rozejść z M1 i M2 są takie same i przy tym samym adresie bazowym BA równy k, mikrorozkazom MA i MB zostanie przypisany ten sam adres k, a dla MC będzie przeznaczona komórka k+1. Sytuacja taka jest niedopuszczalna i wymaga podania na UMA różnych adresów bazowych podczas realizacji mikrorozkazów MA i MB. Oznacza to przypisanie mikrorozkazowi MC dwóch adresów pamięci sterujących.

Do realizacji mikroprogramu o l mikrorozkazach, w większości wypadków konieczna jest pamięć mikroprogramu o pojemności n słów, przy czym $n > l$, czasami dość znacznie. W celu zmniejszenia tej dysproporcji oraz skrócenia czasu potrzebnego do rozmieszczenia mikrorozkazów w PM należy w pierwszej kolejności wyznaczyć adresy mikrorozkazów, których etykiety występują w mikrooperacji rozejścia warunkowego na największą liczbę dróg (mikrooperacja taka wymaga największego spójnego bloku pamięci PM). Na końcu są adresowane mikrorozkazy występujące w mikrooperacjach rozejść bezwarunkowych.

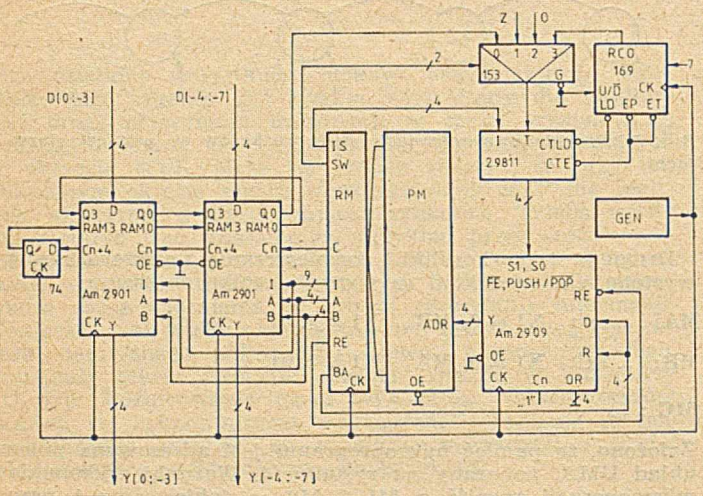
Po zaadresowaniu mikrorozkazów można przystąpić do wypełniania pamięci mikroprogramu. Wymaga to analizy sposobu wykonania mikrooperacji zapisanych w danym mikrorozkazie; odczytania z odpowiednich tabel słów dwójkowych sterujących ich realizacją. Czynność ta jest bardzo pracochłonna i podczas generowania słowa mikrorozkazowego o długości kilkudziesięciu bitów mogą powstać błędy trudne do wykrycia. Dla uproszczenia tej pracy i zwiększenia poprawności jej wykonania opracowano metajęzyki przeznaczone do symbolicznego zapisu zawartości pamięci mikroprogramu. Umożliwiają one konstruktorowi układu mikroprogramowego definiowanie własnych mnemoników, za pomocą których można zastąpić trudne do zapamiętania liczby dwójkowe nazwami oznaczającymi konkretną mikrooperację w MZF. Mikrorozkazy występujące w mikroprogramie są wtedy grupowane ze względu na występujące w nich mikrooperacje. W celu zminimalizowania zapisu definiuje się formaty mikrorozkazów określające wartość sterowań wspólnych dla danej grupy mikrorozkazów oraz podające miejsca wystąpienia pól zmiennych. Pola zmienne są uzupełniane w momencie zapisu zawartości konkretnej komórki pamięci.

Program opisujący zawartość pamięci sterującej ma więc dwie części: część deklaracyjną i część wykonawczą, w której opisuje się pamięć za pomocą wcześniej zdefiniowanych nazw formatów i pól mikrooperacji. Należy zwrócić uwagę na fakt, że może istnieć jedna część deklaracyjna zawierająca pełny opis MZF dla wielu różnych mikroprogramów.

Wśród metaasemblerów najbardziej znany jest AMDASM [6] przygotowany przez firmę AMD. W Instytucie Informatyki opracowano jego rozszerzoną wersję UMAS pracującą pod nadzorem systemu CP/M 2.2.

PRZYKŁAD MIKROPROGRAMOWANEGO ZESPOŁU FUNKCJONALNEGO

Pokazany na rysunku 8 mikroprogramowany zespół funkcjonalny realizuje mnożenie dwóch liczb ułamkowych reprezentowanych słowami, MN, MK: word [0:—7] w kodzie NB.



Rys. 8. Schemat blokowy MZF realizującego mnożenie liczb ułamkowych. Do budowy układu wykorzystano dwa segmenty Am2901

Działanie układu opisuje mikroprogram podany w pierwszej części artykułu. Biorąc pod uwagę mikrooperacje rozejść warunkowych występujące w mikroprogramie oraz możliwości układu Am29811 mikrorozkazy można zaadresować w następujący sposób:

adres 0—M1, 1—M2, 2—M3, 3—M4,
4—M5, 5—M7, 6—M6, 7—M7,

Umieszczenie mikrorozkazu M7 w słowach PM o adresach 5 i 7 zostało spowodowane sposobem sterowania sekwensera Am2909 przez układ Am29811 podczas realizacji rozejść warunkowych. Sterowanie to pozwala warunkowo wybrać źródło adresu D lub PC (F lub PC) przy jednoczesnym wymuszeniu wykonania mikrooperacji L:=DEC(L). Mikrorozkaz M7 (lub M4) musi być więc umieszczony w PM bezpośrednio po M5 i M6. Do sterowania przykładowego MZF wykorzystano funkcje układu Am29811 zamieszczone w tabeli 3.

Tabela 3. Skrócona tabela sterowań układem Am29811

SI [3 : 0]	T	S [1 : 0]	CTLD CTE	Komentarz
1 0 0 1	0	1 1	1 0	ADR ⇐ BA, L: = DEC (L)
	1	0 0	1 1	ADR ⇐ PC
0 0 1 1	0	0 0	1 1	ADR ⇐ PC
	1	1 1	1 1	ADR ⇐ BA
1 1 0 0	X	0 0	0 1	ADR ⇐ PC, L : = 7
0 1 1 1	0	0 1	1 1	ADR ⇐ R
	1	1 1	1 1	ADR ⇐ BA
1 1 1 0	X	0 0	1 1	ADR ⇐ PC

Słowo mikrorozkazowe sterujące działaniem MZF można podzielić na przykład na następujące pola:

S[3:0]=BA[3:0] — adres bazowy dla Am2909,
S[4] = $\overline{RE}$ — sterowanie zapisem do rejestru R w układzie Am2909,
S[8:5]=IS[3:0] — sterowanie układem Am29811,
S[10:9]=SW[1:0] — wybór warunku rozejścia,
S[14:11]=A[3:0] — adres A dla układu Am2901,
S[18:15]=B[3:0] — adres B dla układu Am2901,
S[19] =C0 — przeniesienie wejściowe dla Am2901,
S[28:20]=I[8:0] — sterowanie układem Am2901.

Z powyższego zestawienia wynika, że długość słowa mikrorozkazowego równa jest 29 bitów.

Sterowanie dla przykładowego MZF przedstawiono w tabeli 4. Niektóre sterowania wskazane pod tabelą mają ustaloną wartość. Część sterowań jest identycznych, na przykład S[24]=S[23] lub identycznych, z dokładnością do negacji S[22] = ¬ S[15], S[25] = ¬ S[5]. Zamiast więc pamięci mikroprogramu o organizacji 8 słów 29-bitowych (4 układy SN74188) można użyć pamięci 8 słów 15-bitowych (2 układy SN74188) uzupełnionej dwoma bramkami negacji.

URUCHAMIANIE MIKROPROGRAMOWANYCH ZESPOŁÓW FUNKCJONALNYCH

Uruchamianie MZF dzieli się na dwa etapy:

- funkcjonalne sprawdzenie działania zaprojektowanego układu podczas uruchamiania krok po kroku,
- sprawdzenie własności dynamicznych MZF, w szczególności dobór parametrów sygnału synchronizującego.

W obu tych etapach może zajść potrzeba zmiany zawartości pamięci mikroprogramu, realizowanej najczęściej z układów pamięci stałych PROM. Zapis informacji do pamięci PROM polega na przepaleniu bezpieczników w wybranych komórkach pamięci. Daną komórkę PROM można więc programować tylko raz. Podczas funkcjonalnego uruchamiania MZF zastępuje się układy PROM pamięciami typu UVEPROM o identycznym lub podobnym układzie końcówek — na przykład 2716 zamiast TM624. Unika się wtedy wyrzucania błędnie zaprogramowanych układów PROM, ale konieczny jest ciągły dostęp do programatora pamięci stałych. Oprócz tego, pamięci UVEPROM są kilkakrotnie wolniejsze od układów PROM i mogą nie nadawać się do dynamicznego uruchamiania MZF.

Po krokowym sprawdzeniu działania MZF może okazać się, że dla osiągnięcia interesującego konstruktora stanu uruchamianego układu trzeba wykonać kilkadziesiąt lub nawet kilkaset kroków. Konieczna jest więc możliwość zatrzymywania pracy układu podczas realizacji wybranego mikrorozkazu. Zatem do uruchamiania dużych mikroprogramowanych zespołów funkcjonalnych konieczne jest oprzyrządowanie o następujących własnościach:

- zastąpienie pamięci PROM pamięcią RAM z możliwością łatwej jej modyfikacji,
- śledzenie wykonania mikroprogramu przez sprawdzanie adresów pamięci sterującej, z możliwością zatrzymania działania układu w wybranym mikrorozkazie,
- zbieranie śladu n ostatnio odczytanych z pamięci sterującej mikrorozkazów,
- generowanie sygnału synchronizującego dla uruchamianego układu z możliwością zmiany jego parametrów.

Takimi własnościami charakteryzują się systemy uruchomieniowe mikroprogramowanych układów sterowania. Do

Tabela 4. Sterowania przykładowego MZF

Etykieta mikrooperacji	Adres szesnastkowy	8 7 6	5 4 3	2 1 0	9	8765	5321	09	8765	4	3210
		I [8 : 6]	I [5 : 3]	I [2 : 0]	C	B	A	SW	IS	$\overline{RE}$	BA
M1	0	01X	000	111	0	0000	XXXX	01	1001	X	0000
M2	1	000	000	111	0	XXXX	XXXX	01	0011	X	0001
M3	2	01X	100	011	0	0001	XXXX	XX	1100	0	0111
M4	3	100	011	011	X	0001	XXXX	00	0011	1	0110
M5	4	01X	000	011	0	0001	XXXX	11	1001	1	0011
M7	5	01X	011	011	X	0001	XXXX	10	0111	1	0000
M6	6	01X	000	001	0	0001	0000	11	1001	1	0011
M7	7	01X	011	011	X	0001	XXXX	10	0111	1	0000
		0		1	0	000	0000				0

najbardziej popularnych należy System-29 [6] umożliwiający symulowanie pamięci mikroprogramu o pojemności 2048 × 128b i czasie dostępu ~30 ns.

* * *

W Instytucie Informatyki PW już od 4 lat jest wykorzystywany w zajęciach dydaktycznych i pracach badawczych system SUMUS o pamięci mikroprogramu 4096 × 32b i czasie dostępu ~600 ns [3]. W opracowaniu jest SUMUS II o pojemności pamięci 2048 × 256b i czasie dostępu 100 ns. System SUMUS II będzie wyposażony w 6 sond, których stan z każdego cyklu zegara będzie wyświetlany na ekranie i zapamiętywany w postaci śladu. Sondy będzie można umieszczać w dowolnym punkcie uruchamianego układu i stan ich może być warunkiem zatrzymania działania MZF.

LITERATURA

[1] Budkowski S., Papliński A., Sosnowski J.: Zespoły i urządzenia cyfrowe. Warszawa WNT, 1979

[2] Budkowski S., Pawłowski M.: Problemy projektowania mikroprogramowanych urządzeń cyfrowych z użyciem mikroprocesorów segmentowych. Wydawnictwa Politechniki Warszawskiej 1987

[3] Budkowski S., Woźniak A.: System uruchomieniowy mikroprogramowanych układów sterowania. IV Krajowa Konferencja Naukowo-Techniczna SEP: „Zastosowanie mikroprocesorów w automatyce i pomiarach", Warszawa 13.09.1984

[4] Budkowski S., Dembliński P.: Verification design and description oriented microprogramming language. Proceedings EUROMICRO-78, Munich 1978

[5] Microprogramming Handbook. Advanced Micro Devices 1976

[6] The Am2900 Family Data Book with Related Support Circuits. Advanced Micro Devices 1978

[7] The TTL Data Book for Design Engineers. TEXAS INSTRUMENTS Incorporated 1976.

Konferencje

SIGGRAPH'87

W dniach od 27 do 31 lipca br. w Anaheim (USA, Kalifornia) odbędzie się czternasta, doroczna konferencja i wystawa grafiki komputerowej SIGGRAPH 87 (14 th Annual Conference on Computer Graphics and Interactive Techniques). Organizatorami tej niewątpliwie najbardziej znaczącej na świecie imprezy w omawianej dziedzinie są: sekcja grafiki komputerowej amerykańskiego stowarzyszenia informatycznego ACM (ACM Special Interest Group on Computer Graphics) oraz komitet grafiki komputerowej amerykańskiego stowarzyszenia inżynierów elektryków i elektroników IEEE (IEEE Technical Committee on Computer Graphics).

Wynikający z wieloletnich doświadczeń program tej imprezy jest bardzo rozbudowany i wielopłaszczyznowy. Obejmuje on tzw. Technical Program — klasyczną konferencję poświęconą prezentacji referatów naukowych dotyczących osiągnięć ostatniego roku oraz powszechnie praktykowane dyskusje panelowe (Panel Sessions), zapewniające uczestnikom bezpośredni, aktywny udział w wymianie poglądów wybitnych specjalistów. Następnym, wypróbowanym już w praktyce SIGGRAPH składnikiem imprezy są jednodniowe, siedmiogodzinne seminaria szkoleniowe (Courses), wprowadzające w najważniejsze zagadnienia technologii i zastosowań grafiki komputerowej. Seminaria te uwzględniają trzy poziomy wiedzy uczestników: początkowy, średni i zaawansowany. Ze względu na olbrzymie zainteresowanie w roku ubiegłym, program seminariów został znacznie rozszerzony i obejmuje aż 30 różnych tematów. Tego rodzaju rozszerzenie klasycznego programu konferencji wydaje się być pomysłem godnym naśladowania ze względu na aspekty ogólnospołeczne (przyspieszenie i niewątpliwie podwyższenie poziomu szkolenia, szczególnie istotne w nowych, szybko rozwijających się dziedzinach technologii i zastosowań).

Równocześnie z konferencją i seminariami odbywać się będzie na powierzchni ok. 11 000 m² tradycyjna wystawa najnowszych rozwiązań sprzętowych i programowych grafiki komputerowej z udziałem ponad 250 wystawców.

Podobnie jak w latach ubiegłych, atrakcyjność imprezy podniosą niewątpliwie pokazy filmów oraz z taśm wideo (Film and Video Show), dotyczące wszystkich podstawowych zastosowań grafiki i animacji komputerowej, a także ekspozycja najwybitniejszych dzieł twórczości plastycznej (Art Show) artystów korzystających z coraz większych możliwości technicznych grafiki komputerowej. (WK)

POLSKIE TOWARZYSTWO INFORMATYCZNE
ogłasza
IV OGÓLNOPOLSKI KONKURS
na najlepsze
prace magisterskie z informatyki

W konkursie mogą brać udział wszyscy dyplomanci studiujący na wyższych uczelniach w Polsce (również obywatele innych krajów), których prace dyplomowe dotyczą informatyki i którzy złożyli i obronili te prace w terminie określonym przez regulamin studiów odnośnej uczelni.

Zgłoszenia pracy na konkurs dokonuje autor (autorzy) pracy, przesyłając pod adresem organizatora konkursu:

POLSKIE TOWARZYSTWO INFORMATYCZNE
ODDZIAŁ DOLNOŚLĄSKI
Pl. Grunwaldzki 9 (D-2) p. 15
50-370 **WROCŁAW**

w terminie do dnia 5 października 1987 r. następujące dokumenty:

a) pracę dyplomową,

b) wypełnioną przez autora (autorów) ankietę (formularze ankiet wysłano do dziekanatów i instytutów oraz są także dostępne u organizatorów konkursu),

c) zaświadczenie z uczelni, że zgłoszona na konkurs praca została obroniona jako praca magisterska w roku ogłoszenia konkursu, tj. w okresie od 1 października 1986 do 30 września 1987 r.

Wyniki konkursu będą ogłoszone nie później niż do końca roku 1987.

NAGRODY

I nagroda —30 000 zł
II nagroda —25 000 zł
III nagroda —20 000 zł
trzy wyróżnienia po —10 000 zł

Komisja konkursowa może nie przyznać dowolnej z nagród, bądź podzielić jedną nagrodę pomiędzy kilka prac. Po zakończeniu konkursu przedłożone prace magisterskie zostaną zwrócone autorom.

Zapraszamy do wzięcia udziału w konkursie
Dolnośląski Oddział PTI

JANUSZ ZALEWSKI (oprac.)
Warszawa

Concurrent DOS

System operacyjny Concurrent DOS (po polsku — współbieżny DOS) opracowano w firmie Digital Research Inc. w USA. Jego miejsce w hierarchii mikrokomputerowych systemów operacyjnych przedstawiono na rysunku.

Concurrent DOS można uważać za „nadzbiór" systemów operacyjnych CP/M-86 i PC-DOS, ponieważ działają w nim programy napisane dla obu tych systemów. Jedną z najbardziej korzystnych właściwości systemu Concurrent DOS jest możliwość używania dyskietek zapisanych zarówno w formacie PC-DOS, jak i w formacie CP/M-86. Przy tworzeniu takiego „nadzbioru" konieczny był jednak pewien kompromis, polegający na ogół na pominięciu niektórych cech każdego z systemów pierwotnych.

WSPÓŁBIEŻNOŚĆ

Główna właściwość systemu Concurrent DOS, odróżniająca go od poprzedników, tj. współbieżność, w praktyce jest sprowadzana do wieloprogramowości, gdyż umożliwia wykonywanie czterech programów użytkowych jednocześnie i korzystanie dodatkowo z programu obsługi drukarki, którym jest tzw. zarządca druku (ang. printer manager). Każdemu z programów współbieżnych odpowiada ustalone okno, przez które można komunikować się z tym programem i wydawać mu polecenia.

Concurrent DOS można też uważać za wieloużytkowy system operacyjny (ang. multiuser), ponieważ zapewnia on obsługę dwóch dodatkowych terminali dołączonych przez porty szeregowe COM1 i COM2. Jest jednak oczywiste, że ze względu na ograniczone możliwości tych portów wiele programów korzystających w sposób istotny z właściwości karty sprzęgającej monitora ekranowego nie będzie w takiej konfiguracji działać poprawnie. Odnosi się to głównie do programów adresujących ekran przez bufor w pamięci RAM. Choć terminale dołączone przez porty szeregowe mają również możliwość takiej pracy, ze względu na istotne różnice w budowie adapterów sprzęgających, rzadko kiedy to samo oprogramowanie umożliwia prace zarówno w jednym, jak i w drugim trybie. Wiele programów usługowych, np. procesory tekstowe, arkusze elektroniczne, pomija wywołania do systemu BIOS, adresując bezpośrednio ekran, co jest przyczyną dodatkowych komplikacji i czyni je nieprzenośnymi na inny sprzęt. Inna wadą pracy wieloużytkowej w systemie Concurrent DOS jest znaczny spadek wydajności przy dołączaniu drugiego i trzeciego użytkownika.

W praktyce więc, korzystanie z wieloprogramowości i wieloużytkowości systemu Concurrent DOS podlega wielu ograniczeniom. Najbardziej powszechne zastosowania tego rodzaju polegają na inicjowaniu pracy w trybie wsadowym, np. kompilacji i konsolidacji, nie wymagających interwencji operatora, lub — na wykonywaniu zadań w tle, jak np. obsługa łącza komunikacyjnego, przy jednoczesnym prowadzeniu pracy pierwszoplanowej, np. redagowania tekstu lub modyfikowania arkusza kalkulacyjnego.

ORGANIZACJA SYSTEMU PLIKÓW

Główna różnica między systemami operacyjnymi CP/M-86 i PC-DOS jest odmienna organizacja systemu plików, co stanowiło podstawowa trudność przy próbie ich zintegrowania w systemie Concurrent DOS. Struktura systemu plików każdego z dwóch systemów pierwotnych jest powszechnie znana. Mają one organizację hierarchiczną, lecz tylko PC-DOS zapewnia dowolną głębokość hierarchii, wzorowanej na Unixie, natomiast CP/M-86 ma hierarchię jednopoziomową.

Ponadto, system PC-DOS zapewnia automatyczne umieszczenie datownika pliku w chwili jego utworzenia, co nie występuje w systemie CP/M. Z kolei CP/M umożliwia ochronę plików przez podawanie hasła, czego nie zapewnia PC-DOS. Odpowiednie polecenia systemu Concurrent DOS dopuszczają jednak wykonywanie tych operacji w sposób jednolity. W praktyce więc można przesyłać bez kłopotu pliki z dyskietki w formacie CP/M-86, umieszczonej w jednym napędzie, na dyskietkę w formacie PC-DOS, w drugim napędzie lub odwrotnie.

Dysk stały w systemie Concurrent DOS może mieć strefę (ang. partition) przeznaczoną na pliki w standardzie PC-DOS i drugą strefę na pliki w standardzie CP/M-86. Odpowiednie strefy mają wówczas logiczne nazwy "C:" i "D:", a dla drugiego dysku stałego "E:" i "F:". W systemie Concurrent DOS istnieją tzw. pseudodyski (ang. floating drives), nazwane "N:" i "O:", którym można przypisać dowolny dysk fizyczny, strefę lub skorowidz (systemu PC-DOS), co jest o tyle użyteczne, że każde odwołanie do pseudodysku powoduje przeszukiwanie przypisanego mu skorowidza. Można utworzyć także dysk elektroniczny w pamięci RAM, "M:", działający ok. 10 razy szybciej od dysku stałego.

OPIS GŁÓWNYCH POLECEŃ

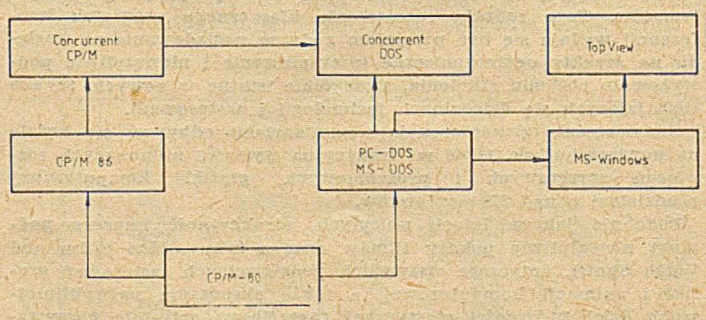
Część poleceń obowiązująca w systemie Concurrent DOS została przeniesiona albo z systemu PC-DOS, jak BATCH, CD, COPY, MD, RD itp., albo z systemu CP/M-86, jak PIP, USER (niektóre z nich są wspólne dla wszystkich systemów, np. DIR, ERASE, RENAME, TYPE itp.). Te polecenia różnią się bardzo niewiele od swych pierwowzorów.

Najciekawsze w systemie Concurrent DOS są nowe polecenia, charakterystyczne dla zawartej w nim współbieżności. Wszystkie są wykonywane techniką menu, tzn. z przedstawionym na ekranie wykazem dopuszczalnych czynności i rozjaśnieniem aktualnie wybranej nazwy.

Operowanie plikami

Do operowania plikami służy polecenie FM wywołujące program odgrywający rolę zarządcy plików (ang. file manager). Zarządca plików dzieli ekran na trzy części, tzw. panele:

- **panel poleceń** (ang. command panel), zawierający wykaz poleceń możliwych do wykonania w tym trybie,
- **panel obiektowy** (ang. object panel), zawierający wykaz plików lub innych obiektów, którymi operuje się za pomocą poleceń,



Miejsce systemu Concurrent DOS w hierarchii mikrokomputerowych systemów operacyjnych

● **panel podpowiedzi** (ang. prompt panel), zawierający opis aktualnie wybranego polecenia i opis poleceń przypisanych klawiszom funkcyjnym F1—F10.

W menu polecenia FM znajdują się wszystkie inne polecenia służące do operowania plikami, tzn. TYPE, PRINT, COPY, RENAME, DELETE, BACKUP, EDIT, FORMAT, operowania skorowidzami i in. Praca w tym trybie jest o wiele łatwiejsza niż wydawanie poleceń metodą tradycyjną (tzn. tekstowo), stosowaną w systemach PC-DOS i CP/M.

Operowanie oknami

Operowanie oknami jest najistotniejszą cechą systemu Concurrent DOS. Podstawowym poleceniem służącym do operowania oknami jest WINDOW. Ustanawia się nim cztery okna, przez które można kierować wykonywaniem programów współbieżnych. Wyboru jednego z okien dokonuje się jednoczesnym przyciśnięciem klawisza CTRL i jednego z klawiszy 1—4 klawiatury numerycznej, choć możliwe jest także zdefiniowanie własnego przyporządkowania. Przyciśnięcie klawiszy CTRL-DEL umożliwia przełączenie wyświetlania okna z całego ekranu na jego uprzednio zdefiniowany fragment.

Poszczególne warianty tego polecenia umożliwiają wyświetlenie (WINDOW VIEW) aktualnych wartości parametrów każdego okna (początkowy wiersz i kolumna, liczba wierszy i kolumn, kolory itp.) oraz ich zmienianie (WINDOW CHANGE), a także uaktywnienie określonego okna (WINDOW TOP), wyświetlenie okna na całym ekranie (WINDOW FULL) i zapisanie zawartości okna (ekranu) na plik dyskowy (WINDOW WRITE).

Wszystkie funkcje tego polecenia można wykonywać techniką menu, a więc interakcyjnie, przy użyciu zarządcy okien (ang. window manager) o nazwie WMENU. Instaluje się go wywołaniem:

A > WMENU
Window Manager Installed

a wywołuje przyciśnięciem klawiszy CTRL i "+".

Operowanie listami menu

W systemie Concurrent DOS każdemu menu odpowiada specjalny kontrolny plik z danymi (lub jego część) o domyślnej nazwie MENU.DAT, zawierający treść poleceń, które są interpretowane, i tekst wyświetlany przez polecenie wbudowane RUNMENU. Do tworzenia plików kontrolnych menu służy specjalny edytor EDITMENU.

Jest to bardzo ważne narzędzie, dzięki któremu można tworzyć własne, służące różnym celom menu, znacznie przyspieszające i ułatwiające pracę. Oczywiście, samo przygotowanie menu odbywa się również techniką menu.

Do kopiowania plików menu, zagęszczania ich zawartości itp., służy dodatkowe polecenie COPYMENU.

Zarządzanie drukowaniem

Zarządzanie procesem prowadzenia wydruków jest wykonywane przez specjalny program, zwany zarządcą druku (ang. printer manager), wywoływany poleceniem PRINTMGR START.
Zarządca druku może operować w systemie Concurrent DOS jako piąty program współbieżny, oprócz programów zainicjowanych z czterech okien. Zarządcy plików można przyporządkować aż 5 drukarek (trzy dołączone przez porty równoległe i dwie — przez porty szeregowe), lecz można też używać drukarek przyporządkowanych tylko poszczególnym oknom. W każdym oknie można używać drukarki przypisanej poleceniem:

PRINTER = n

gdzie **n** oznacza numer drukarki (od 0 do 4). Ten sam wydruk może być wyprowadzony jednocześnie na kilka drukarek.

Poszczególne opcje polecenia PRINTMGR oznaczają:

PRINTMGR HELP — samouczek elektroniczny,

PRINTMGR PRINT nazwy_plików parametry — drukowanie zawartości plików w formacie określonym przez parametry,

PRINTMGR STATUS — wyświetlenie aktualnego stanu prac w kolejce do drukowania,

PRINTMGR DELETE n — zatrzymanie procesu drukowania nr **n** bez możliwości kontynuacji,

PRINTMGR SUSPEND — wstrzymanie drukowania i zachowanie aktualnego stanu kolejki na dysku (co umożliwia późniejsze kontynuowanie tak wstrzymanego wydruku). Każde z tych poleceń można oczywiście wywołać techniką menu, co dotyczy także samego programu PRINTMGR, wywoływanego z menu HDMENU lub FDMENU.

INNE POLECENIA

Kilka innych poleceń systemu Concurrent DOS ma dość ciekawe właściwości, o których warto krótko wspomnieć.

Polecenia BACK i REST służą odpowiednio do tworzenia rezerwowych (ang. backup) kopii plików i sprowadzania ich (ang. restore) z powrotem na dysk stały, według schematu zapisanego na pliku CONTROL.BR. Ponieważ programy BACK i REST używają kilku innych plików z rozszerzeniem nazwy BR, wyklucza się użycie takiego rozszerzenia w nazwach plików użytkowych. Zależnie od celu tworzenia kopii rezerwowej lub ponownego sprowadzenia pliku na dysk można utworzyć kilka plików CONTROL, dodając do nazwy każdego z nich jeden znak: CONTROLnBR. Odpowiednie wywołania mają wówczas postać:

BACK CONTROL=n

lub

REST CONTROL=n

Interesujące możliwości zapewnia polecenie CARDFILE, udostępniające i utrzymujące elektroniczną książkę adresową. Inne polecenia, a właściwie całe złożone programy, DREDIX i DRTALK, umożliwiają odpowiednio — redagowanie tekstów oraz komunikację przez linie telefoniczne. Ich opis jest jednak dość złożony i przekracza ramy tego artykułu.

Szereg prostych poleceń zapewnia ustawianie parametrów systemowych. Przykładowo:

8087 = ON | OFF

informuje o użyciu przez program koprocesora arytmetycznego. Polecenie:

ADDMEN = n

służy do rozszerzenia obszaru pamięci dla programów typu EXE, a polecenie:

COMSIZE = n

do przypisania wielkości pamięci programowi typu COM. Z kolei, polecenie:

FUNCTION plik.PFK

służy do przyporządkowania tekstu klawiszom funkcyjnym i ich kombinacjom z klawiszami sterującymi: CTRL, ALT, SHIFT itp. Polecenie:

ORDER ext1, ext2, ext3, ext4

zmienia kolejność przeszukiwania plików wykonywalnych, z domyślnej (COM, EXE, CMD, BAT) na podaną w tym poleceniu. Inne polecenie:

SYSDISK dysk: / ścieżka *

podobne do znanego z systemu PC-DOS polecenia PATH, służy do ustanawiania ścieżki w systemie plików, przeszukiwanej w wypadku, gdy żądany plik nie znajduje się w skorowidzu bieżącym. Polecenie:

SETPORT parametry

ustanawia tryby pracy portów szeregowych w sposób zbliżony do polecenia MODE dla systemu PC-DOS.
Najważniejsze polecenie ustanawiania parametrów systemowych, SETUP, umożliwia najrozmaitsze zmiany domyślnych właściwości systemu Concurrent DOS, np. ustanowienie dysku elektronicznego, przypisanie portów itp. Używa się go przede wszystkim stosując technikę menu.

* * *

Concurrent DOS w wersji 4.1 zajmuje ok. 150 KB pamięci operacyjnej i umożliwia wykorzystanie pozostałego obszaru

pamięci na programy lub dysk elektroniczny, lecz tylko do pojemności 640 KB.

Niektóre programy dostarczane z systemem Concurrent DOS nie współpracują z konsolą dołączoną przez port szeregowy, np. FM, DRTALK, DREDIX, EDITMENU, RUNMENU, DSKMAINT, HDMAINT (dwa ostatnie służą do formatowania i weryfikacji dysków). Inne programy, jak FUNCTION, PRINTMGR, SETPORT i SETUP, współpracują z portem szeregowym tylko w wersji tekstowej, lecz nie w wersji menu.

Inną wadą związaną z pracą wielołużytkową jest fakt, że niektóre programy, np. BASICA, przeprogramowują porty szeregowe, co wyklucza ich użycie w środowisku wielołużytkowym. Choć ich stosowanie w zestawie jednołużytkowym jest dopuszczalne, to nie mogą w nim współistnieć dwa programy korzystające jednocześnie z interpretera BASICA.

W drugiej części artykułu omówię dokładniej ważniejsze polecenia związane ze stosowaniem techniki menu.

---

*) Ze względu na brak w drukarni znaku ukośnika (ang. backslash) w tekście artykułu zastąpiono go znakiem dzielenia „/".

Janus/Ada — użyteczne narzędzie do nauczania Ady

Ada jest językiem przyszłości Departamentu Obrony Stanów Zjednoczonych (DOD, Department of Defense). Pomimo że DOD wydał zarządzenie, aby wszyscy jego kontrahenci używali Ady od początku 1985 roku, to brak dostępnych i sprawdzonych kompilatorów opóźnił jej szerokie zastosowanie. Z poglądem, że Ada jest najlepszym językiem programowania można się nie zgadzać, jednak niezależnie od tego Ada będzie językiem preferowanym w zastosowaniach rządowych. DOD ma nadzieję, że stosując jeden standardowy język zredukuje koszty eksploatacji oprogramowania.

Właściciele mikrokomputerów, którzy chcą nauczyć się Ady, będą jednak zawiedzeni, bo dostępne kompilatory są albo realizacjami częściowymi albo niestandardowymi podzbiorami pełnego języka Ada. Janus/Ada firmy RR Software (w wersji 1.4.7) jest także niestandardowym podzbiorem Ady dla systemów operacyjnych MS-DOS i CP/M-80. Janusowi brakuje większości cech, które wyróżniają Adę spośród innych języków wysokiego poziomu, a ponadto ma on pewną liczbę cech niestandardowych. Jednakże Janus/Ada jest użytecznym narzędziem do nauczania właściwości złożonych języków programowania.

Niniejszy opis dotyczy wersji Janus/Ada na komputer IBM PC dla systemu operacyjnego MS-DOS. Ada była pierwotnie zaprojektowana do zastosowań w czasie rzeczywistym, jak kierowanie pociskami lub przetwarzanie danych radarowych. Dostęp do pocisków kierowanych nie jest łatwy, a na dodatek Janus nie realizuje wielozadaniowości Ady, więc kompilator można sprawdzić, badając jedynie przydatność Ady jako języka uniwersalnego. Ze względu na to, że w Janus/Adzie nie ma wbudowanych bibliotek graficznych, do sprawdzenia napisano prostą, tekstową grę przygodową.

Podobną grę przygodową w języku BASIC pisze się w ciągu dziesięciu godzin. Programowanie gry w Janus/Adzie trwa dłużej, być może z powodu braku doświadczenia w używaniu tego języka. Jednakże program wynikowy jest lepiej zbudowany, łatwiejszy do zrozumienia i modyfikacji. Znający Pascal powinni łatwiej uczyć się Ady niż ci, którzy zdobyli doświadczenie programując w innych, gorzej zbudowanych językach. Podobieństwo między językami może być jednak mylące. Linia niedozwolona w Janus/Adzie przypomina poprawny kod w Pascalu, w związku z tym lokalizowanie błędu może trwać długo.

Do kompilatora jest dołączonych kilka programów w Janus/Adzie, przetłumaczonych z Pascala. Żaden z tych programów nie zasługuje na uwagę, jednak pokazują one, jak zrealizowano pewne funkcje.

NIEKTÓRE ODSTĘPSTWA OD NORMY

Janus nie jest, ściśle biorąc, podzbiorem Ady. Problem, który sprawia szczególne trudności, dotyczy wywołań parametrycznych. Standardowa Ada pozwala wywołać jakąś funkcję lub procedurę, która ma przyjąć domyślne wartości parametrów, przez odwołanie do jej nazwy. Janus — podobnie do wcześniejszej wersji Ady — wymaga, aby dodać pusty zbiór parametrów w nawiasach; tak więc zamiar użycia parametrów domyślnych jest wyrażony jawnie.

Janus/Ada nie używa standardowych napisów Ady, czyli nie ma w niej prostego sposobu wczytania napisu przy użyciu następującej procedury Ady:

```
get (slowo);
```

lub

```
get_line (slowo);
```

Janus/Ada wyklucza używanie napisów w procedurze **get**. Trzeba więc używać funkcji **get_line** zamiast procedury **get**. Wskutek tego każdy program, który korzysta z wejścia-wyjścia, jest w Adzie niestandardowy. Aby przeczytać napis, trzeba wywołać funkcję **get_line**:

```
slowo:=get_line( );
```

Wymagane są tu nawiasy.

Aby upodobnić gotowy program do znormalizowanej Ady, można utworzyć proste procedury w celu ukrycia tych niestandardowych wywołań. Jeżeli kompiluje się taki program przy użyciu kompilatora pełnej Ady, to trzeba wymienić tylko te procedury.

Niestandardowy sposób obsługi tablic również stwarza pewne kłopoty. Można zrealizować samodzielnie pewne brakujące właściwości Ady, ale wtedy traci się trochę na jej elegancji. Na przykład, poprawne w Adzie przypisanie:

```
y(1...10):=x(1..10);
```

kopiuje każdy element **x(i)** do odpowiadającego mu elementu **y(i)**. W Janus/Adzie tak się nie stanie, ponieważ Janus/Ada nie realizuje wycinków tablic ani napisów. Znaczy to, że nie ma dostępu do podzbiorów elementów tablicy. Jeśli **x** i **y** nie są tablicami złożonymi z napisów, to powyższą instrukcję można zastąpić następującą:

```
for i in 1..10 loop
   y(i) := x(i);
end loop; —— for i
```

Jeśli **x** i **y** są napisami, to sprawa jest trudniejsza. Kilka niestandardowych funkcji i procedur operowania napisami opisano w dodatku do podręcznika. Aby wykonać dokładnie wymagane przypisanie, należy użyć następujących instrukcji:

```
y := extract(y,11,length(y));
insert(y,extract(x,1,10),1);
```

Pierwsza instrukcja usuwa 10 początkowych znaków wektora **y**, a druga wstawia pierwszych 10 znaków wektora **x** na początek **y**. Nie jest to jednak tak proste jak poprawne w Adzie przypisanie tablic.

PROCES KOMPILACJI

Dowolny program w Janus/Adzie można łatwo zawrzeć w oddzielnym segmencie i oddzielnie skompilować. Postępując w ten sposób, kilku programistów może pisać program niezależnie, gdyż każdy zna nazwy i parametry podprogramów, które piszą inni. Dowolne zmiany wprowadzone później do tych podprogramów będą wymagać tylko powtórnej kompilacji uzależnionych segmentów.

Kompilator wykonuje cztery oddzielne przebiegi, podczas których na ekran trafia bardzo dużo informacji. Dla typowego użytkownika większość z nich jest jednak bezużyteczna. Kiedy kompilator znajduje błąd, wyświetla się linia programu, w której on wystąpił, oraz linia poprzednia wraz z jej numerem, a także — odpowiedni komunikat.

Błędy wykonania są bardziej kłopotliwe. Kiedy pojawia się błąd wykonania, na ekranie wyświetla się jedynie komunikat i numer linii. Jeżeli edytor tekstu nie używa numerów linii, to aby znaleźć błąd, należy samemu policzyć linie, co jest niełatwym zadaniem, gdy błąd jest, na przykład, w linii 675.

Każda kompilacja trwa od dwóch do pięciu minut, zależnie od długości pliku i od tego, czy kompilowany plik jest tylko specyfikacją, czy zawiera wykonywalny program. Długie pliki mogą być podzielone na części do oddzielnej kompilacji. Jest to użyteczne, kiedy prosta procedura musi być wielokrotnie skompilowana w trakcie uruchamiania. Po skompilowaniu wszystkich segmentów można dołączyć program główny i utworzyć plik typu COM. Podobnie jak większość innych kompilatorów, kompilator Janus/Ady tworzy pliki typu COM dłuższe od programów źródłowych, ponieważ są do nich dołączone podprogramy biblioteczne.

PRZYKŁADY

Janus/Ada nie jest zoptymalizowanym kompilatorem ani też nie optymalizuje kodu, który wytwarza. Znając jego cenę i szybkość wprowadzenia na rynek trudno się temu dziwić. Jednakże kompilator wymagałby istotnych ulepszeń przed użyciem do opracowywania oprogramowania na sprzedaż. Znany program wzorcowy SIEVE (sito Erastotenesa) napisany w Adzie został skompilowany w czasie 184,7 s, połączony w 15,1 s i wykonany w 29,4 s. Kompilatory większości, jeśli nie wszystkich, innych języków dla IBM PC tworzą efektywniejszy kod i robią to szybciej.

```
package body FLOATBCH is
  CONST1 : constant FLOAT := 3.141597E0;
  CONST2 : constant FLOAT := 1.7839032E4;
  COUNT  : constant INTEGER := 1000;
  A,B,C  : FLOAT;
  I : INTEGER;
begin
  A := CONST1;
  B := CONST2;
  for I in 1..COUNT loop
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
    C := A * B;
    C := C / A;
  end loop;
  PUT ("Wykonana"); NEW_LINE;
end FLOATBCH;
```

Wydruk 1. Program testujący operacje zmiennoprzecinkowe

Program testujący operacje zmiennoprzecinkowe (wydruk 1) został skompilowany w czasie 184 s, połączony w 15,8 s i wykonany w 2,6 s. W tym wypadku czas wykonania był krótszy niż w wypadku kilku kompilatorów języka C, mimo że szybkość kompilacji była względnie mała. Należy pamiętać przy tym, że zastosowano koprocesor 8087 i że Janus/Ada może używać liczb zmiennoprzecinkowych na IBM PC tylko wtedy, gdy komputer jest wyposażony w ten koprocesor; nie ma programowanej realizacji arytmetyki zmiennoprzecinkowej.

Program wzorcowy do obliczania liczb Fibonacciego nie wykonałby się po przetłumaczeniu, ponieważ Janu/Ada nie przyjmuje 16-bitowych liczb całkowitych bez znaku; powodują one błąd wykonania, gdy przekroczą dopuszczalną długość. Jeżeli przeredaguje się program stosując typ **long_integer** Janus/Ady, to pojawia się przepełnienie stosu, ponieważ kompilator używa tylko 64 KB pamięci na dane (na program używa innych 64 KB). Programy Quicksort i IOfile używane do testowania kompilatorów również korzystają z długich liczb całkowitych, więc nie przeprowadzono testów przy ich użyciu.

```
package body FIBO is
  NTIMES : constant INTEGER := 10;       -- krotnosc  wyliczenia
                                            wartosci Fibonacciego
  NUMBER : constant INTEGER := 24; -- mozna dzialac na liczbach
                                                  16- bitowych
  VALUE  : INTEGER;
  I      : INTEGER;
  ------------------
  function FIB(X: in INTEGER) return INTEGER is
  begin
    if X > 2 then
      return (FIB(X - 1) + FIB(X - 2));
    else
      return 1;
    end if;
  end; -- funtion FIB
  -------------------
begin -- FIBO
  PUT(NTIMES);
  PUT("ITERACJE:");
  NEW_LINE;
  for I in 1..NTIMES loop
    VALUE := FIB(NUMBER);
  end loop; -- for i
  PUT("Fibonacci(");
  PUT(NUMBER);
  PUT(") = ");
  PUT(VALUE);
  NEW_LINE;
end; -- FIBO
```

Wydruk 2. Program wzorcowy dla ciągu Fibonacciego wyrażony w standardowej Adzie

Nie jest łatwo nauczyć się korzystania z typu **long_integer** w Janus/Adzie, nawet po kilku telefonach do RR Software. W podręczniku stwierdzono, że **long_integer** jest typem standardowym, ale w rzeczywistości trzeba używać oddzielnego pakietu bibliotecznego zwanego LONGOPS. Kopie pakietów bibliotecznych znajdują się na dysku dostarczanym wraz z kompilatorem.

```
with LONGOPS;
package body FIBO is
  use LONGOPS;
  NTIMES : constant INTEGER := 10; -- krotnosc wyliczenia
                                      wartosci Fibonacciego
  NUMBER : constant LONG_INTEGER := LINT(24);    -- mozna
                          dzialac na liczbach 16-  bitowych
  ONE    : constant LONG_INTEGER := LINT(1);
  TWO    : constant LONG_INTEGER := LINT(2);
  VALUE  : LONG_INTEGER;
  I      : LONG_INTEGER;
  ----------------------
  function FIB(X: in LONG_INTEGER) return LONG_INTEGER is
  begin
    if LGT(X,TWO) then
      return LADD(FIB(LSUB(X,ONE)),FIB(LSUB(X,TWO)));
    else
      return ONE;
    end if;
  end; -- function FIB
  --------------------
begin -- FIBO
  PUT(NTIMES);
  PUT("Iteracje:");
  NEW_LINE;
  for I in 1..NTIMES loop
    VALUE := FIB(NUMBER);
  end loop; -- for i
  PUT("Fibonacci(");
  PUT(L_TO_INT(NUMBER));
  PUT(") = ");
  PUT(L_TO_INT(VALUE));
  NEW_LINE;
end; -- FIBO
```

Wydruk 3. Program wzorcowy dla ciągu Fibonacciego wyrażony w Janus/Adzie z użyciem typu long_integer

Z liczbami typu **long_integer** nie można postępować w Janus/Adzie tak, jak z liczbami typu integer, ponieważ zasadniczo należą one do typów definiowanych przez użytkownika; dodawanie lub konwersja typów musi być wykonywana przy użyciu jednej z funkcji bibliotecznych pakietu LONGOPS. W rezultacie, program używający liczb typu **long_integer** w Janus/Adzie wygląda zdecydowanie inaczej od programu używającego liczb typu integer w

bardziej standardowej wersji języka. Wydruk 2 przedstawia program Fibonacciego w standardowej Adzie; powstaje w nim błąd przepełnienia, ponieważ 24 liczba Fibonacciego jest 16-bitową liczbą bez znaku, a Janus/Ada umożliwia użycie tylko 15-bitowych liczb całkowitych bez znaku albo 16-bitowych ze znakiem. Wydruk 3 przedstawia ten sam program w Janus/Adzie przekształcony z użyciem typu **long_integer**; powstaje w nim przepełnienie stosu, wywołane wielokrotną rekurencją i żądaniem użycia dużego obszaru pamięci danych.

DOKUMENTACJA I ROZWÓJ

Podręcznik Janus/Ady ma postać zgodną z podręcznikiem normy języka. Każda część odzwierciedla odpowiedni rozdział normy i omawia różnice między Janusem a Adą. W podręczniku ostrzega się, że nie wyczerpuje on tematu i sugeruje czytelnikowi, aby korzystać dodatkowo z normy.

Autorzy opisu Janusa odwołują się do normy Ady z 1980 roku, która nie jest już aktualna. W procesie recenzowania przez ANSI (American National Standards Institute) wprowadzono do języka zmiany i prawdziwa Ada została opisana w normie wydanej w styczniu 1983 roku.

Do podręcznika dołączono dość rzetelny spis haseł (skorowidz), ale pominięto niektóre pozycje. Na przykład, przy pisaniu programów obsługujących napisy łatwo zauważyć, że w skorowidzu nie ma odwołań do części 15, która zawiera spis funkcji operujących napisami. W podręczniku są też pewne błędy, np. stwierdzenie, że **long_integer** jest typem standardowym, a faktycznie tak nie jest.

Centrum sterowania rozwojem Ady AJPO (ang. Ada Joint Program Office) nalega, aby każda częściowa realizacja Ady była wyraźnie nazwana jako odbiegająca od normy, a wszystkie standardowe cechy języka, których brakuje danemu kompilatorowi, były jasno określone. Firma RR Software dołącza do dokumentacji spis zrealizowanych i niezrealizowanych cech Ady. Niektóre z ważniejszych brakujących cech języka wymieniono w tabeli.

O ile brakujące cechy Janus/Ady mają wpływ na jej mniejszą przydatność dla programistów, to niestandardowa realizacja innych cech może sprawiać istotne trudności.

Spis niektórych niezrealizowanych lub niestandardowych cech Janus/Ady

Cecha	Różnica
Wycinki	Dopuszcza się jedynie odwołania do wycinków tablic lub napisów, np.: a (1..5)
Napisy	Niezgodne z normą Ady (zmienna długość)
Parametry nazwane i domyślne	Dopuszcza się domyślne parametry wejściowe lub parametry nazwane w wywołaniu podprogramu, np.: ATTACK (ENEMY => SAM, WAPON => KNIFE); ATTACK (ENEMY => FRED); procedure ATTACK (ENEMY: in PERSON: = = DAVE; WEAPON: in TOOLS: = GKN);
Zadania	Brak wielozadaniowości
Wyjątki	Brak obsługi wyjątków

Rodzajowość	Można łatwo przedefiniować podprogramy z użyciem typów danych, np.:
	procedure EXCHANGE (u, v: in out ELEM) is
	t: ELEM;
	begin
	t: = u; u: = v; v: = t;
	end EXCHANGE;
	procedure SWAP is new EXCHANGE (CHARACTER);
	procedure SWAP is new EXCHANGE (ELEM => => INTEGER);

Niestandardowa obsługa napisów i plików oznacza, że trzeba będzie przepisać prawie wszystkie programy w Janus/Adzie w celu przetłumaczenia ich przy użyciu kompilatora prawdziwej Ady.

Do chwili obecnej firma RR Software wprowadziła na rynek następne wersje Janus/Ady i poinformowała, że w kompilatorze wprowadzono wiele rozszerzeń i zmian. Mimo że Janus/Ada nie jest pełną realizacją Ady i brakuje mu wielu jej standardowych cech, to jest narzędziem niedrogim i użytecznym dla tych, którzy chcą nauczyć się tego języka, zanim zakupią kompilatory prawdziwej Ady na mikrokomputery.

Oprac. JERZY STRYCHARCZYK
na podst. BYTE, nr 2, 1985

NOWE KSIĄŻKI

R. Forsyth, R. Rada: Machine Learning — Applications in Expert Systems and Information Retrieval. Ellis Horwood, 1986.

Książka zawiera omówienie przykładowych zastosowań maszynowego uczenia się, które stanowi aktywnie rozwijającą się dziedzinę sztucznej inteligencji. Mocną stroną książki jest jej praktyczne podejście do zagadnień, które dotąd były wyłączną domeną badaczy. Pierwsza część książki jest szerokim wprowadzeniem do metod uczenia maszynowego, które omówiono w odniesieniu do budowy systemów ekspertowych. Drugą część poświęcono dokładnemu przedstawieniu koncepcji inteligentnego systemu wyszukiwania informacji wykorzystującego mechanizmy uczenia się.

A. Kandel: Fuzzy Mathematical Techniques with Applications. Addison-Wesley, 1986.

Książka przedstawia podstawowe pojęcia teorii zbiorów rozmytych w kontekście praktycznych zastosowań, które ze względu na niejednoznaczny charakter nie poddają się bezpośrednio analizie za pomocą konwencjonalnych środków matematycznych. Omówiono takie zastosowania jak: przetwarzanie wiedzy w systemach ekspertowych, struktury relacyjnych baz danych, identyfikowanie wzorców (ang. pattern clustering), modelowanie niepewności i drzewa przeszukiwań.

J. J. Craig: Intruduction to Robotics — Mechanics and Control. Addison-Wesley, 1986.

Książka stanowi kompendium wiedzy z dziedziny manipulatorów mechanicznych. Zawiera szczegółowy opis kinematyki dwóch popularnych robotów przemysłowych, wiele przykładów dwu- i trójprzegubowych manipulatorów oraz analizę podstawowych zagadnień związanych z budową robotów, popartą przykładami i zadaniami programistycznymi.

Ian C. Pyle: Ada. Seria BIO, nakład 10 000 egz., 284 str., cena 300 zł, Warszawa, 1986.

Jest to pierwsze wydanie książki opisującej jeden z najnowszych języków programowania — Ada, stanowiący znaczny krok naprzód w technologii programowania. Zgodnie z podstawowym celem książki, materiał przedstawiono w niej w sposób zakładający znajomość programowania. Taki sposób przedstawienia języka różni się od jego „suchej" definicji. Ma to ułatwić wyjaśnienie podstawowych pojęć programistom, którzy pragną nauczyć się Ady, nie stając się specjalistami od pisania kompilatorów. Książka jest przeznaczona dla projektantów wbudowanych systemów komputerowych, pracowników nauki oraz dla studentów kierunków informatycznych. Angielski oryginał uzupełniono w polskim przekładzie dodatkami na temat środowiska programowego Ady i systemu CAMAC.

(M.M.)

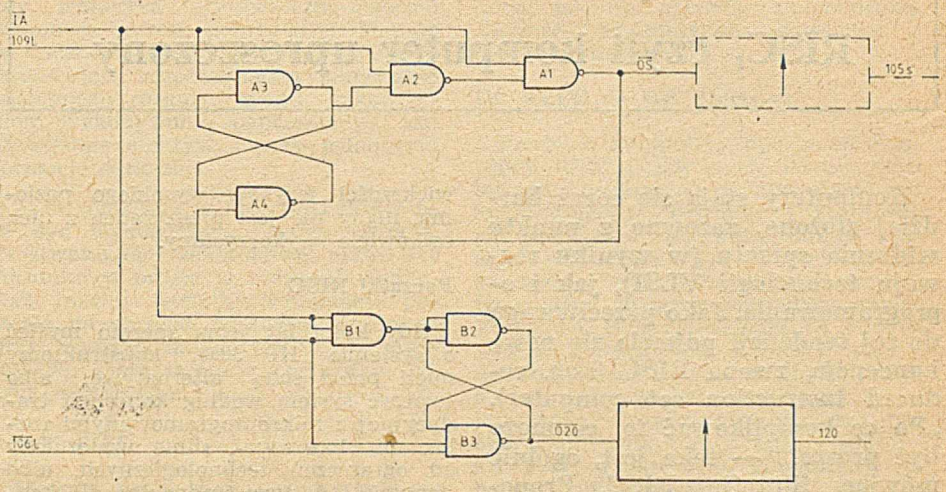
Układ pracy naprzemiennej dla DZM 180-KSRE

Terminal typu DZM 180-KSRE oraz współpracujący z nim adapter UPD 305-8/5, w fabrycznym wykonaniu, są przystosowane do pracy dwukierunkowej jednoczesnej. W wypadku użycia modemów 600/1200 wymusza to pracę na telefonicznych liniach trwałych czteroprzewodowych. W związku z brakiem linii telefonicznych, w ZETO Białystok opracowano i sprawdzono układy przystosowujące terminal i adapter do pracy naprzemiennej na linii dwuprzewodowej (rys. 1 i 2).

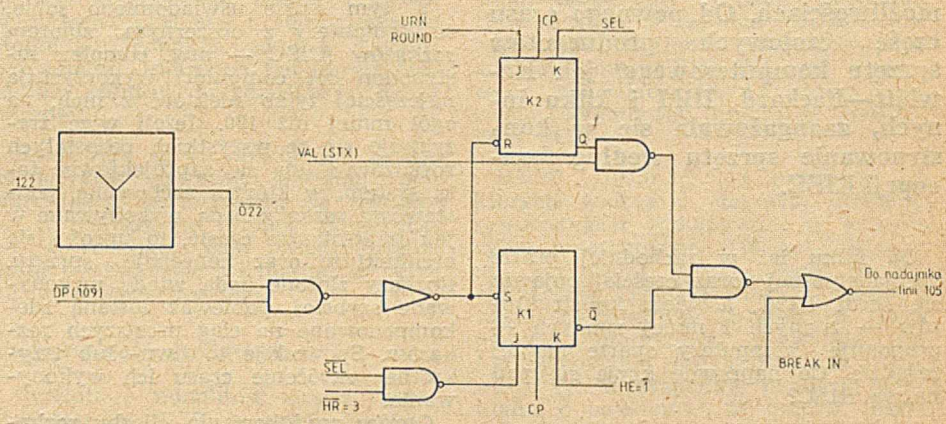
Nazwy sygnałów zewnętrznych są zgodne z dokumentacją techniczną, np.: sygnał IA w adapterze określa kierunek transmisji (IA=1 — transmisja wyjściowa). W układzie wykorzystano linie kanału powrotnego (120, 122), dzięki czemu adapter UPD zyskuje wyższy priorytet dla transmisji w kierunku KSRE.

Jest to istotne, gdy poziom sygnału na linii 105 od strony terminala nie zostanie zmieniony przed wysterowaniem sygnału IA w adapterze (rys. 3). W DZM 180-KSRE linią 105 sterują przerzutniki K1 i K2. Pierwszy z nich służy w zasadzie do przesyłania znaku STX (przerwania) i znaku potwierdzenia (NAK lub ACK). Przed przesłaniem tekstów należy użyć klawisza TURN ROUND, również w programach nie wykorzystujących transmisji TURN ROUND.

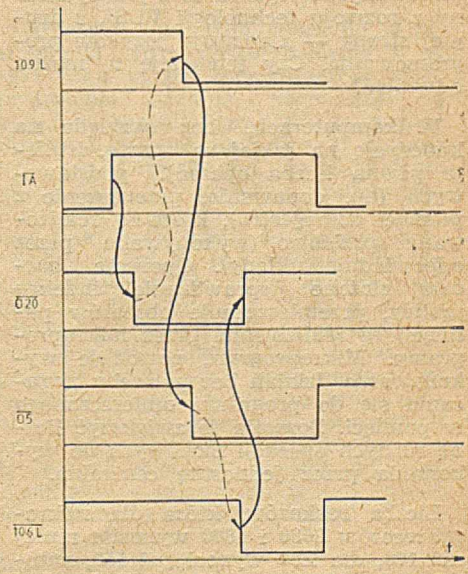
STANISŁAW WYSZYŃSKI
ZETO Białystok



Rys. 1. Schemat ideowy dodatkowego układu w adapterze UPD



Rys. 2. Przykładowe rozwiązanie układu w DZM180-KSRE



Rys. 3. Przebiegi czasowe w układzie adaptera

RISC, czyli komputer uproszczony

Komputery stają się coraz bardziej złożone, zarówno z punktu widzenia sprzętu (w wyniku rozwoju technologii VLSI), jak i oprogramowania. Jako przeciwwaga do tej tendencji pojawia się nowa koncepcja, zwana RISC (ang. reduced instruction set computer). „Po co komplikować to, co może być proste?" — taka jest, ogólnie mówiąc, filozofia RISC. Proponuje ona komputery bardziej uniwersalne, szybsze i o większych możliwościach. Od pewnego czasu część czołowych producentów sprzętu komputerowego, jak Hewlett—Packard, IBM i kilku innych, zaangażowała się w konstruowanie sprzętu według koncepcji RISC.

Od kilku lat w zachodniej prasie specjalistycznej coraz częściej pojawia się skrót RISC, a wielu konstruktorów — znanych i mniej znanych — proponuje komputery oparte na tej zasadzie. Co naprawdę kryje się pod nazwą RISC?

Rozwój komputerów czyni sprzęt i oprogramowanie coraz bardziej skomplikowanymi. Wynika to z jednej strony z rozwoju technologii VLSI, z drugiej strony — z rozwoju języków programowania. Czy tendencję tę można odwrócić?

W komputerach, bez względu na koncepcję na jakiej są oparte, istnieje pewna liczba operacji podstawowych (np. dodawanie, przenoszenie z pamięci do rejestru), które są realizowane sprzętowo (wykonywane przez odpowiednie układy). Operacje bardziej złożone (sprawdzanie indeksu tablicy, przemieszczanie obszarów pamięci) są realizowane przez mikroprogramy. Mikroprogram powoduje wykonywanie działań bez potrzeby zwracania się do programu umieszczonego w pamięci. Pozwala to zwiększyć liczbę różnych operacji możliwych do wykonania przez jednostkę centralną.

Liczba rozkazów, która dla mikroprocesorów 8080 i 6800 wynosiła mniej niż 100, dla Z80 wynosi już 158, natomiast w mikrokomputerze VAX 11/780 wzrasta do ponad 300. Jeżeli jeszcze uwzględnić różne sposoby adresowania i różne typy danych, to liczba rozkazów na poziomie języka asemblera znacznie się zwiększa.

Powoduje to wzrost wymagań w stosunku do kompilatora każdego języka programowania oraz wydłuża, zamiast skracać, czas wykonywania programu. W dodatku złożony zbiór rozkazów, mający zaspokoić wymagania większości języków wysokiego poziomu, nigdy nie jest przez żaden z nich całkowicie wykorzystany.

Początki RISC

Gdy kilka lat temu zaczęto myśleć o systemie HP 3000, konstruktorzy mieli przed sobą alternatywę: albo tworzyć system według koncepcji tradycyjnej z mikroprogramowanymi rozkazami bazowymi, silnie uzależniony od ograniczeń technologicznych, albo porzucić strukturę tradycyjną dla koncepcji RISC, która może stworzyć nowe możliwości rozwoju.

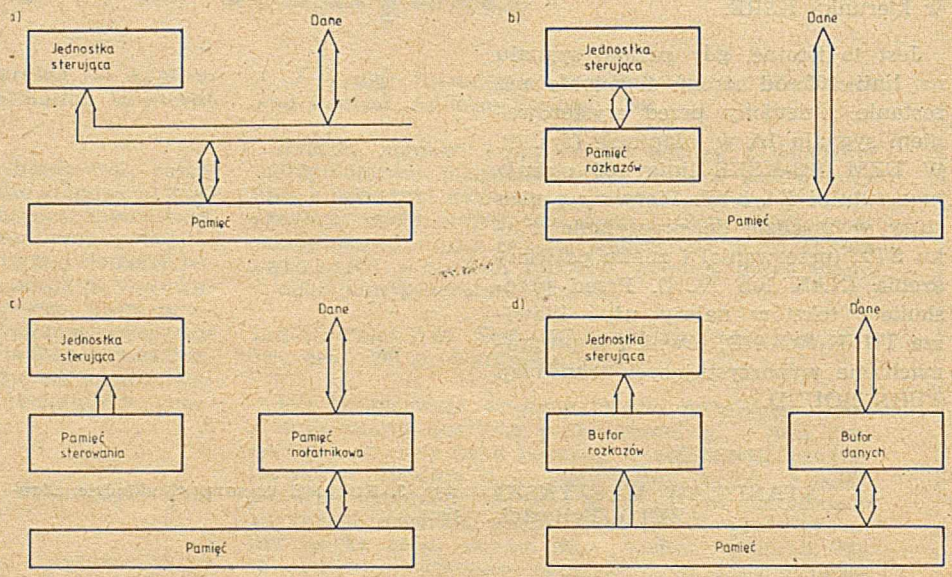
W tym czasie uświadomiono sobie, że komputery z obszernym zbiorem rozkazów (CISC — ang. complex instruction set computer) wykorzystują najczęściej tylko niektóre z nich, na ogół mniej niż 100. Jeżeli więc zrezygnuje się ze wszystkich pozostałych rozkazów i uda się zaprojektować taki system, w którym zachowane, podstawowe rozkazy będą wykonywane w jak najkrótszym czasie, to uprości się architekturę oraz koncepcję sprzętu. Rozkazy złożone będą wtedy wykonywane szybciej, ponieważ zostaną zdekomponowane na ciąg prostszych rozkazów. Spowoduje to dwu- lub trzykrotne skrócenie czasu ich wykonywania.

Oprócz zredukowania liczby rozkazów koncepcja RISC wpływa na inne istotne cechy systemów komputerowych:

- im mniejsza jest liczba rozkazów, tym mniej bitów potrzeba do ich zapisania; pozwala to skrócić czas wykonywania programu,
- każdy rozkaz jest wykonywany w jednym cyklu zegarowym,
- rozkazy mają jednolity format, najczęściej długość 32 bitów ze stałymi polami; kod operacji i obszary adresowania rejestrów zajmują zawsze te same pozycje; przesłania z pamięci i do pamięci są realizowane najczęściej za pomocą jednego trybu adresowania, co upraszcza dekodowanie rozkazów, a w konsekwencji ułatwia pisanie kompilatorów,
- zredukowanie przestrzeni fizycznej zajętej przez mikrokod pozwala tworzyć układy o większym zagęszczeniu, przesyłanie danych jest realizowane już nie między układami, a wewnątrz jednego układu, co znacznie zwiększa szybkość wykonywania operacji.

Rejestry

Zmniejszenie liczby rozkazów nie jest celem, lecz tylko środkiem. Nie chodzi bowiem o stworzenie maszyny zdolnej do szybkiego wykonywania rozkazów, która nie będzie miała dużej pamięci wirtualnej, większość czasu będzie bezczynnie czekała na przyjęcie informacji z pamięci czy też będzie ograniczona przez mechanizmy wejścia-wyjścia. Hierarchia pamięci i systemy wejścia-wyjścia są więc istotnymi elementami w tworzeniu całego systemu.



Rys. 1. Ewolucja architektury komputerów od von Neumanna do RISC:
a) pierwsze komputery (architektura von Neumanna) mają bardzo mało rejestrów. Szybkość logiki i dostępu do pamięci, prawie jednakowe w komputerach lampowych, stały się różne w chwili, gdy miejsce lamp zajęły tranzystory, a później układy scalone;
b) wprowadzenie mikrokodu, wykorzystujące nierównowagę między logiką i pamięcią główną;
c) wprowadzenie bufora lub pamięci notatnikowej pozwala na szybki dostęp do pamięci głównej o dużej pojemności, ale bardzo wolnej;
d) szyna danych została uproszczona, aby zmniejszyć długość cyklu oraz kodowanie rozkazów w taki sposób, aby nie powodować zbędnych rozszerzeń

Początkowo w koncepcji RISC kładziono nacisk na różnicę między operacjami mikrokodowymi i sprzętowymi, ale równie ważne w tej koncepcji jest dobre wykorzystanie rejestrów głównych zamiast mikrokodu. Zysk z takiego podejścia jest oczywisty — rozkazy przesyłania między rejestrami są dużo szybsze od przesyłania opartego na dostępie do pamięci centralnej.

Rejestry mogą być uniwersalne, co upraszcza ich wykorzystanie przez kompilator. Kompilator analizuje dane, kontroluje ich przepływ, może też zwalniać niepotrzebne rejestry.

W niektórych komputerach opartych na koncepcji RISC, jak np. Pyramid Technology, zawierających ponad 500 rejestrów, są one zorganizowane w stos, którego elementy są przydzielane i zwalniane dynamicznie.

Projekt RISC z Uniwersytetu w Berkeley (stan Kalifornia) wykorzystuje koncepcję „okna rejestrów". Przesuwanie adresów wirtualnych odbywa się przez przemieszczanie okien rejestrów. Okna te częściowo pokrywają się, co powoduje, że jedna grupa rejestrów zapewnia powiązanie z dwoma oknami. Ten mechanizm pozwala uniknąć kopiowania danych z jednego rejestru do drugiego.

Okna rejestrów mogą być także używane do przekazywania parametrów do procedur.

Hierarchia pamięci

W koncepcji RISC jednym z najważniejszych punktów jest hierarchizacja pamięci, ponieważ funkcje zapisu i odczytu pamięci są dużo wolniejsze niż funkcje logiczne.

Następnym, po zbiorze rejestrów, poziomem hierarchii jest pamięć notatnikowa. Tworzy ona bufor między pamięcią centralną i procesorem. Ma niewielką pojemność, ale bardzo szybki dostęp, co powoduje, że czas dostępu do pamięci zaczyna być porównywalny z czasem wykonywania operacji przez jednostki arytmetyczno-logiczne.

W architekturze RISC pamięć notatnikowa jest widoczna dla oprogramowania, a zarządzanie hierarchią pamięci odbywa się w sposób jawny. Pozwala to na zmniejszenie czasu potrzebnego na przesyłanie adresów wirtualnych, pobieranie rozkazów oraz pobieranie i zapamiętywanie danych.

Architektura RISC proponowana przez firmę Hewlett-Packard może mieć 2^{32} przestrzeni adresów wirtualnych o długości 2^{32} bajtów każda.

Można używać równocześnie dwu pamięci notatnikowych, co pozwala na równoczesne pobieranie danych i rozkazów.

Szybki dostęp do pamięci, dzięki użyciu rejestrów, pozwala na podział operacji na niezależne porcje, tak aby komputer pracował zakładkowo (np. pobierał następną instrukcję, w czasie, gdy jednostka arytmetyczno-logiczna wykonuje jeszcze poprzednią). Metoda ta, zwana zakładkową (ang. pipeline), pozwalająca rozpoczynać wykonywanie następnego rozkazu przed zakończeniem wykonywania poprzedniego, jest szczególnie efektywna przy jednolitym formacie rozkazów.

Tym sposobem w każdym cyklu może być wykonywany jeden rozkaz. Przetwarzanie zakładkowe może być stosunkowo proste w realizacji, ponieważ rozwój techniki kompilacji pozwala na obróbkę rozkazów złożonych przy wykorzystaniu tych cykli zegarowych, które w innym wypadku byłyby nieaktywne.

Języki RISC

Można powiedzieć, że architektura RISC nie zawiera mikrokodu albo też, że ma wyłącznie mikrokod. Może być on zapisany za pomocą języka wysokiego poziomu i standardowego otoczenia. Jednakże, zamiast specjalnej pamięci, mikrokod ten używa hierarchii pamięci przydzielanej dynamicznie.

Koncepcja RISC doskonale pasuje do języków wysokiego poziomu pod warunkiem, że instrukcje są regularne, a realizowane funkcje symetryczne. W rezultacie całe programowanie może odbywać się wirtualnie w języku wysokiego poziomu.

Uniwersalność architektury

Prostota i jednolitość zbioru rozkazów maszyn RISC powodują również, że wszystkie komputery stają się kompatybilne, co w wypadku architektury CISC nigdy nie jest możliwe.

Ograniczony zbiór rozkazów pozwala na ich implementację na poziomie sprzętu. Nie nakłada to żadnych ograniczeń technologicznych — do implementacji architektury nadają się zarówno układy TTL i NMOS, będące podstawą obecnych modeli Hewlett-Packarda, jak i CMOS oraz ECL.

Przeciwnicy koncepcji RISC wysuwają jednak następujące argumenty:

- programy pisane dla maszyn RISC są dużo dłuższe niż przy posługiwaniu się złożoną listą instrukcji; zysk z bezpośredniego wykonywania instrukcji może być zbyt mały w porównaniu ze stratami spowodowanymi koniecznością zastępowania złożonych instrukcji zbyt długim ciągiem instrukcji prostych,
- w architekturze RISC nie jest możliwe wykonywanie niektórych specyficznych instrukcji języków wysokiego poziomu,
- do niektórych ciągów instrukcji, które w innym wypadku byłyby mikroprogramowane, trzeba dostawać się za pomocą wywołania podprogramów.

Większość tych obaw jest jednak bezpodstawna. Kompilator często będzie zdolny do tworzenia programów równie krótkich, jak rozwiązania konwencjonalne. W innym wypadku słuszne będzie dołożenie pewnych układów, aby skrócić niektóre złożone operacje, jak np. operacje zmiennoprzecinkowe, arytmetyczne dziesiętne i operacje macierzowe.

Od RTPC do HP Precision

Pierwsze komputery oparte na koncepcji RISC pojawiły się na początku lat osiemdziesiątych na uniwersytetach amerykańskich (Berkeley i Stanford), ale prace nad projektami takich maszyn rozpoczęto już ponad 10 lat wcześniej, zwłaszcza w laboratoriach firmy IBM w centrum badawczym w Yorktown Heights. Chodzi tu o projekt 801, który miał stworzyć maszynę konkurencyjną dla serii IBM 370, o dwa razy większej szybkości przetwarzania. Projekt ten przybrał postać materialną na początku 1986 roku jako system 6150, zbudowany na bazie mikroprocesora RTPC (RT — RISC Technology). Zawiera on bardzo szybki 32-bitowy procesor RISC, system operacyjny będący pochodną Unix System V, system zarządzania zasobami, urządzenia wejścia-wyjścia kompatybilne z IBM PC oraz koprocesor emulujący PC/AT, który pozwala wykonywać programy rodziny IBM PC.

RTPC „rozumie" 118 rozkazów, z których większość stanowią operacje rejestr-rejestr wykonywane w jednym cyklu maszynowym (170 ns). Jedynymi rozkazami zwracającymi się do pamięci są LOAD i STORE. Procesor używa 16 rejestrów 32-bitowych. Dwie pamięci notatnikowe pozwalają na funkcjonowanie mechanizmu zakładkowego. RTPC dysponuje pamięcią wirtualną o pojemności 2^{40} bajtów. Średnia szybkość przetwarzania wynosi ok. 2 mips (milionów instrukcji na sekundę).

Wcześniej niż IBM maszynę RISC stworzyła firma Hewlett-Packard. W 1981 roku grupa konstruktorów przeprowadziła badania, które stały się punktem wyjścia projektu o nazwie Spectrum. Zamierzenie to jest ogromnym przedsięwzięciem, mobilizującym ok. tysiąca inżynierów i naukowców. Chodzi o stworzenie systemu, a raczej tylko jego jądra, do którego można by dobudowywać różne układy specjalistyczne. Ta nowa architektura, nazwana HP Precision, musi być niezależna od technologii i implementacji, powinna nadawać się do współpracy z systemami programowanymi w językach wysokiego poziomu, oraz być zdolna do emulacji poprzednich architektur Hewlett-Packarda (w tym także podsystemów wejścia-wyjścia). Architektura otrzymana w ten sposób jest zgodna z koncepcją RISC.

Inne projekty RISC

W 1982 roku na uniwersytecie w Berkeley (stan Kalifornia) stworzono, pod kierunkiem Davida Pattersona i Carlosa Sequin, procesor RISC I jako układ NMOS zawierający 31 kodów operacji i jeden tryb adresowania. Model ten, z powodu błędów koncepcji, został zastąpiony przez RISC II — układ scalony zawierający 41 tysięcy tranzystorów, mogący wykonywać 39 operacji i zawierający dużą liczbę rejestrów.

Od tego czasu wielu konstruktorów zaczęło tworzyć maszyny według koncepcji RISC:
- RIDGE 32 (BULL SPS 9) wykonany ze standardowych układów TTL, rozpoznaje 98 kodów operacji, wykonywanych w większości w jednym cyklu zegarowym, ale zawiera tylko 32 rejestry,
- PYRAMID TECHNOLOGY 98, zawierający dwa procesory TTL (superminikomputer), pracuje w trybie zakładkowym, dysponuje pamięcią notatnikową 4 KB, zawiera 528 rejestrów 32-bitowych zgrupowanych w 16-poziomowy stos.

Nad projektami maszyn RISC pracują również następujące firmy: Digital, która zamierza stworzyć system jednoprocesorowy o szybkości przetwarzania 15—20 mips, ATT, Acorn, która proponuje mikroprocesor 32-bitowy nazwany ARM (Acorn RISC Machine), oraz Thomson, rozwijająca 64-bitowy system wieloprocesorowy SCQM, który ma być ukończony w latach dziewięćdziesiątych.

RISC i przetwarzanie równoległe

Koncepcja RISC nie przekreśla dotychczasowych osiągnięć w dziedzinie rozwoju sprzętu komputerowego. Stosowana do sekwencyjnego przetwarzania zakładkowego nadaje się ona również do przetwarzania równoległego.

Przykładem pogodzenia architektury równoległej i RISC może być Transputer Inmos (IMS T242). Ta 32-bitowa maszyna, oparta na koncepcji współbieżności, ma zredukowaną liczbę rozkazów, a rozkazy podstawowe (+, —, AND, OR, itp.) są wykonywane w jednym cyklu maszynowym (50 ns). Przewidziany dla tej maszyny język Occam jest przystosowany zarówno do przetwarzania równoległego, jak i do komunikacji. Transputer, podobnie jak inne komputery RISC, może być programowany również w językach wysokiego poziomu, takich jak Pascal, Fortran czy C.

Compute Engine firmy Mentor Graphics łączy w sobie architekturę równoległą, przetwarzanie zakładkowe i ograniczony zbiór rozkazów z jednym rozkazem wykonywanym w ciągu jednego cyklu zegarowego. Upraszcza to bardzo problem synchronizacji, właściwy maszynom o architekturze równoległej.

Compute Engine wykorzystuje przetwarzanie równoległe polegające na poszukiwaniu operacji wspólnych dla całego oprogramowania, np. mnożenia, dodawania, generowania adresów. Wszystkie takie operacje są wykonywane równocześnie w jednym mikroprocesorze. Osiąga się w ten sposób szybkość przetwarzania do 10 mips. Pamięć RAM o pojemności 20 MB pozwala zrezygnować z zarządzania pamięcią wirtualną.

* * *

Architektura RISC jest nie tyle realizacją nowej myśli, co raczej syntezą postępu technologii półprzewodnikowej oraz lepszego zrozumienia roli kompilatora i systemu operacyjnego. W ten sposób spełnienie założeń koncepcji RISC wydaje się być jedną z decydujących cech nowych, szybszych i lepszych komputerów.

DOROTA INKIELMAN

EUROMICRO'86

Mikroarchitektury, rozwój i zastosowania

W dniach 15—18 września 1986 roku, w malowniczej scenerii uniwersytetu Ca'Foscari w Wenecji, odbyło się kolejne, dwunaste sympozjum EUROMICRO. Organizatorom nadesłano 121 prac z 30 krajów, z czego zaakceptowano do prezentacji 71, dzieląc je na 19 różnych sesji. Równolegle do trzech podstawowych sesji odbywała się sesja komunikatów, więc na konferencji wygłoszono łącznie ponad 100 referatów. Zostały wygłoszone również trzy tzw. wykłady zaproszone.

Pierwszy z nich, pt. „SUPRENUM — system wieloprocesorowy typu MIMD dla złożonych obliczeń naukowych", wygłosił prof. Ulrich Trottenberg z GMD (RFN). SUPRENUM jest superkomputerem do zastosowań numerycznych, budowanym we współpracy kilkunastu instytucji w RFN. Kolejny wykład, nt. „Porównanie przyczyn występowania błędów w systemach cyfrowych", wygłosił prof. Edward McCluskey ze Stanford (USA). Trzeci wykład, o nieco odmiennym od poprzednich charakterze, traktujący o porównaniu przemysłów układów półprzewodnikowych Europy, Japonii i Stanów Zjednoczonych, wygłosił wiceprezydent firmy Motorola, dr Pasquale Pistorio.

Przechodząc do krótkiego omówienia zasadniczego programu konferencji, chciałbym zwrócić uwagę na znamienny moim zdaniem fakt rosnącego udziału prac na tematy CAD/CAT/CAM. Już pierwsza sesja była tego dowodem. Przedstawiono na niej referaty dotyczące różnych aspektów zastosowania symulacji jako techniki CAD. W pierwszym referacie pt. „Obliczenia sterowane zdarzeniowo jako metoda szybkiej symulacji projektu cyfrowego", W. Hahn (RFN) przedstawił koncepcję równoległego specjalizowanego komputera do szybkiej symulacji złożonych systemów cyfrowych, który wykorzystuje m.in. doświadczenia z realizacji komputerów sterowanych danymi. Kolejnym referentem był Paolo Prinetto z Politechniki w Turynie. Jego wystąpienie dotyczyło wykorzystania języka opisu złożonych relacji czasowych TPDL w środowisku symulatora współbieżnego.

Na sesji poświęconej wbudowanym systemom komputerowym (ang. embedded systems) zaprezentowano interesujące przykłady zastosowań. Wskazano na trudności związane z uruchamianiem tych systemów oraz zaproponowano metody, które mogą ten proces w istotny sposób usprawnić.

Trzecia sesja dotyczyła nowej, pasjonującej grupy zastosowań mikrokomputerów, tzw. sztucznej inteligencji (ang. artificial intelligence — AI), a dokładniej — systemów komputerowych, które wykorzystują techniki AI. Dwie pierwsze prace były poświęcone jednemu z głównych problemów realizacji systemów AI, a mianowicie zwiększeniu ich efektywności przez częściowo sprzętową implementację translatorów języków programowania logicznego. W pierwszym referacie, B. Knoedler i W. Rosenstiel (RFN) zaproponowali preprocesor Prologu oparty na MC 68000 i matrycach bramek. Druga praca dotyczyła nowych struktur obliczeniowych dla efektywnej interpretacji Lispu w systemach mikrokomputerowych (E. van Puttkamer, W. Eicher, RFN).

Jedną z sesji, którą zorganizowano po raz pierwszy, była sesja nt. kompilacji krzemowej (ang. silicon compilation). Ten nieco przewrotny termin, zaproponowany w 1979 roku przez D. Johannsena, ma oznaczać (automatyczne) generowanie (lub transformowanie) projektu topologii układu scalonego, ze specyfikacji użytkownika napisanej w języku wysokiego poziomu. Jakkolwiek w ramach tej sesji nie zaprezentowano jeszcze kompilatora spełniającego wszystkie wymagania, to prace przedstawiające jego elementy, są godne uwagi. Najbardziej zaawansowanym systemem CAD, zaprezentowanym na tej sesji, był system oparty na języku KARL III, autorstwa R. Hartensteina z Kaiserslautern (RFN).

Sesja nt. lokalnych sieci komputerowych zgromadziła prace, których autorzy omawiali zagadnienia: środowiska programowego dla sieci komunikacyjnych, wyboru odpowiedniego protokółu dla sieci oraz metodologii projektowania układów komunikacyjnych.

Sesja nt. systemów wieloprocesorowych jest tradycyjnie jedną z mocniejszych stron konferencji EUROMICRO. W tym roku zaprezentowano na niej

m.in. nowy układ VLSI, wspomagający monitorowanie komunikacji w systemach wielomikroprocesorowych ze wspólną pamięcią (R. Klar, N. Luttenberger, RFN), analizę modeli pamięci notatnikowej w systemach wieloprocesorowych oraz system mikroprocesorowy realizujący Modulę-2 z dzieloną pamięcią dla zastosowań do sterowania w czasie rzeczywistym (E. Debaere i J. M. van Campenhout, Belgia).

Środowiska programowe były wiodącym zagadnieniem kolejnej sesji, podczas której mówiono m.in. o narzędziach i technikach rozproszonego uruchamiania projektów oraz o środowiskach umożliwiających uruchamianie architektur równoległych.

Szczególnie interesująca była sesja nt. syntezy wysokiego poziomu (ang. high level synthesis). Zaprezentowano na niej m.in. różne metody syntezy wspomaganej komputerem, gdzie wejściowe specyfikacje projektanta stanowiły język programowania współbieżnego Occam (G. Collins, M. Edwards, Wielka Brytania) i sieć Petriego (Z. Peng, K. Kuchciński, Szwecja). P. Marwedel (RFN), autor jednego z najbardziej znanych systemów automatycznego projektowania zwanego MIMOLA, przedstawił po raz pierwszy algorytmy syntezy zastosowane w tym systemie. Z kolei autorzy z Twente (grupa L. Spaanenburga) przedstawili system CAD dla układów VLSI, wykorzystujący techniki kompilacji i modularyzacji stosowane w językach programowania wysokiego poziomu. Ostatni referat tej sesji dotyczył metod optymalizacji w syntezie wysokiego poziomu.

Na kolejnej sesji poświęconej systemom tolerującym uszkodzenia zaproponowano prace poświęcone: diagnostyce na poziomie systemowym (F. Lombardi, USA), nowej, tolerującej uszkodzenia architekturze dla dyskretnej transformacji Fouriera (A. Antola, Włochy) oraz przemysłowej aplikacji systemu tolerującego uszkodzenia (J. van Gennip, Holandia).

Problemy generowania testów były na konferencji prezentowane w trzech referatach. W pierwszym z nich A. Behbahani i F. Hill (Iran) przedstawili inteligentny automatyczny system testowania układów sekwencyjnych SCIRTSS, sterowany językiem programowania sprzętu AHPL. Drugi (F. Distante, Włochy) stanowił propozycję programowego narzędzia automatycznego generowania testów. W trzecim referacie doc. Krzysztof Sapiecha z Politechniki Warszawskiej zaprezentował nową metodę generowania testów dla układów opisanych proceduralnym językiem opisu sprzętu.

Na sesji zatytułowanej „Narzędzia uruchomieniowe" mówiono o testowaniu innego rodzaju obiektów, a mianowicie oprogramowania systemów czasu rzeczywistego. T. Bemmerl (RFN) zaprezentował taki właśnie system, uzasadniając jego przydatność dla systemów wbudowanych z mikroprocesorami 16- lub 32-bitowymi. B. Lazzarini i C. Prete Włochy przedstawili z kolei architekturę interakcyjnego systemu DISEB przeznaczonego do uruchamiania programów dla systemu wieloprocesorowego, stanowiącego węzeł w architekturze typu MARA firmy Selenia. Problem monitorowania wydajności systemu wieloprocesorowego został poruszony w referacie F. Gregoretti z Politechniki w Turynie, natomiast S. Das Gupta (USA) zreferował pewną metodę symulacji systemów wieloprocesorowych.

Zagadnienia sprzętu oraz oprogramowania układów VLSI omawiano na odrębnej sesji. M. Ansorge ze Szwajcarii przedstawił metodologię projektowania indywidualnych mikroprocesorów typu RISC. W kolejnych wystąpieniach omówiono: implementację rejestrów przesuwnych z liniowym sprzężeniem zwrotnym jako układu VLSI (F. Neri, Włochy), realizację szybkich pamięci dla mikroprocesorów w technologii CMOS (IBM, RFN) oraz system projektowania topologii układu scalonego (P. Frison, Francja).

Bardzo złożone problemy projektowe układów VLSI wymagają zastosowania różnych narzędzi programowych i sprzętowych. Jednym z niezbędnych elementów zintegrowanego systemu komputerowo wspomaganego projektowania jest odpowiedni język opisu i projektowania sprzętu. Znaczenie tej klasy języków stale się zwiększa. W ostatnich latach zaproponowano ich bardzo wiele, wyłoniła się więc potrzeba opracowania standardu w tej dziedzinie. Z dwóch głównych propozycji w tym zakresie, tj. rodziny języków CONLAN oraz języka VHDL, na konferencji EUROMICRO dyskutowano o tym pierwszym. H. Eveking z Darmstadt mówił o formalnie wyprowadzonym w ramach CONLAN-u języku SMAX, natomiast. P. Prinetto zaprezentował rozszerzenie bazowego CONLAN-u o reprezentację zależności czasowych. Interesującą propozycją nowego języka w tej dziedzinie jest język „H" G. Carlsteda ze Szwecji. Dwa ostatnie referaty sesji nt. języków opisu sprzętu dotyczyły reprezentacji graficznej projektu (J. Kivelae, Finlandia) oraz semantyki języka opisu sprzętu (T. Larson, Szwecja).

Brak miejsca nie pozwala na omówienie wszystkich sesji, m.in. bardzo interesującej moim zdaniem sesji komunikatów. Wymienię więc tylko tytuły pozostałych sesji zasadniczego programu konferencji:

- systemy rozproszone,
- organizacja pamięci masowych,
- architektury specjalizowane,
- narzędzia inżynierii oprogramowania,
- systemy automatyki przemysłowej,
- wydajność systemów komputerowych.

Dokładne streszczenia wszystkich sesji oraz teksty wygłoszonych na konferencji komunikatów zostaną zamieszczone w najbliższym numerze czasopisma: Microprocessing and Microprogramming — The EUROMICRO Journal.

Pragnąłbym jeszcze zwrócić uwagę na fakt, iż materiały z konferencji EUROMICRO staną się znacznie bardziej dostępne w kraju dzięki nowej polityce wydawniczej tej organizacji. Mianowicie począwszy od bieżącego roku materiały konferencyjne będą wydawane jako kolejny tom EUROMICRO Journal, który wszyscy subskrybenci będą otrzymywali przed konferencją. Tegoroczne materiały zostały wydane jako tom 18, numery 1—5.

Kolejna konferencja EUROMICRO odbędzie się w dniach 14—17 września 1987 roku w Portsmouth, w Wielkiej Brytanii.

dr inż. **ADAM PAWLAK**

Komputery optyczne

W prasie fachowej coraz częściej pojawiają się informacje o zaawansowaniu prac nad optycznym przetwarzaniem sygnałów. Wszystkie publikacje podkreślają ogromny potencjał nowej technologii. Obliczenia będą wykonywane z prędkością światła, czyli co najmniej o dwa rzędy wielkości szybciej niż przewidywana maksymalna szybkość przetwarzania elektronicznego. Soczewki optyczne będą realizować pewne obliczenia matematyczne, które są bardzo trudne do przeprowadzenia na drodze cyfrowej. Optyczna holografia umożliwi trójwymiarowe gromadzenie informacji o niespotykanej dotąd gęstości zapisu.

Komputery optyczne przetwarzają informację zakodowaną w wiązkach światła. Można wyodrębnić kilka rodzajów komputerów optycznych. W **cyfrowych komputerach optycznych** wykorzystuje się nieliniowe lub dwustabilne materiały optyczne w sposób analogiczny do wykorzystania tranzystorów w komputerze elektronicznym. **Analogowe komputery optyczne** opierają się na ciekawych właściwościach soczewek, które umożliwiają szybkie wyznaczanie transformaty Fouriera oraz splotu. W pewnych rozwiązaniach stosuje się układy systoliczne (ang. systolic arrays), które zapewniają dużą dokładność obliczeń dzięki analogowemu przetwarzaniu danych cyfrowych. **Optyczne systemy rozpoznawania obrazów** wykorzystują technologię optyczną, w celu przyspieszenia procesu wyznaczania funkcji korelacji.

Z wyjątkiem pewnych urządzeń radiolokacyjnych (np. radar aperturowy) oraz kilku zastosowań w sektorze wojskowym, komputery optyczne nie posiadają jak dotąd walorów komercyjnych. Technologia optyczna jest jednak na tyle zaawansowana, że w wielu ośrodkach działają już prototypowe systemy obliczeń analogowych oraz systemy przetwarzania obrazów. W 1986 roku zostały zademonstrowane co najmniej dwa prototypy analogowych komputerów optycznych. Intensywne prace badawcze są prowadzone w wielu firmach amerykańskich (DuPont, GTE, IBM, Lockheed, Motorola, Sperry, 3M, Texas Instruments, Honeywell, ATT, Westinghouse, General Dynamics i in.). Wielkie korporacje przemysłowe są wspomagane przez takie ośrodki akademickie jak: Carnegie-Mellon, University of California, Caltech, MIT i Stanford. Najaktywniejszymi krajami europejskimi w tej dziedzinie są Wielka Brytania, Belgia, RFN, Włochy i Francja. Intensywne prace są prowadzone również w Japonii.

Jedna z niedawno przeprowadzonych analiz zawiera następujący scenariusz rozwoju komputerów optycznych:

1986 — prototyp uniwersalnego komputera optycznego,

1987 — prototyp superkomputera optycznego o możliwościach większych od możliwości superkomputerów elektronicznych,

1988 — tania pamięć optyczna,

1989 — komercyjny układ przestrzennej modulacji światła o dużej szybkości i rozdzielczości,

1990 — scalenie elementów dwustabilnych w mikroukładzie,

1990 — optyczna pamięć asocjacyjna,

1990 — komercyjny macierzowy procesor optyczny i hybrydowy komputer elektroniczno-optyczny,

1991 — praktyczna możliwość optycznego łączenia krzemowych obwodów scalonych,

1995 — pierwszy kompletny komputer optyczny.

W. MACHURA
na podst. Computer Graphics and Applications, wrzesień 1986

Zmarł profesor Fred Margulies

Fred Margulies urodził się w 1917 roku w Wiedniu, gdzie studiował mechanikę na Uniwersytecie Technicznym, a w latach 1947—1951 pracował jako inżynier. W 1951 roku zaczął działalność w związkach zawodowych, gdzie do 1980 roku zajmował wiele stanowisk. Fred Margulies szczególnie interesował się automatyzacją, przetwarzaniem informacji i ich społecznym oddziaływaniem. Tak więc, prof. Margulies był urzędnikiem związkowym, socjologiem i informatykiem.

W wieku 42 lat powrócił do Uniwersytetu Technicznego, gdzie uczestniczył w pracach komisji zajmującej się zagadnieniami nowoczesnych technik obliczeniowych. Zorganizował wiele sympozjów, stanowiących wspólne przedsięwzięcia Uniwersytetu i związków zawodowych. Dużym wydarzeniem w tej działalności była konferencja IFIP pt. „Human Choice and Computers", która odbyła się w Wiedniu, w 1974 roku. Bez tej konferencji nie powstałby nigdy Komitet Techniczny TC9 IFIP ds. społecznych aspektów przetwarzania informacji. Dzięki Fredowi Margulliesowi ten temat tak szybko stał się przedmiotem miedzynarodowej współpracy.

Od 1972 roku aż do śmierci, prof. F. Margulies piastował kilka stanowisk w Międzynarodowej Federacji Sterowania Automatycznego (IFAC), był również członkiem założycielem i członkiem rady wykonawczej Austriackiego Towarzystwa Komputerowego.

W IFIP, począwszy od założenia TC9, prof. Margulies był jego wiceprzewodniczącym, a w 1982 roku, w czasie wyjątkowych trudności, przejął na jeden rok obowiązki przewodniczącego. Fred Margulies był również bardzo aktywnym członkiem Grupy Roboczej IFIP WG9.1 (Computers and Work).

Spis publikacji prof. Margulliesa obejmuje 30 pozycji. Wykładał na Politechnice Wiedeńskiej od 1975 roku aż do śmierci. Prezydent Austrii mianował go profesorem. Fred Margulies był nauczycielem nie tylko dla młodszych studentów — własnym przykładem uczył wszystkich współczesnych.

Prof. Margulies zmarł niespodziewanie 10 lutego 1986 roku. Dziedzina przetwarzania informacji i ruch związkowy straciły wyjątkową osobowość, a wielu innych — prawdziwego przyjaciela, który osiągnął zdumiewająco wiele, jednocząc ludzi. Pamięć o Nim zostanie zachowana, a jego spuścizna będzie nam towarzyszyć przez wiele lat.

(MK)

Komputerowe projektowanie systemów baz danych

Ostatnio bardzo dużym zainteresowaniem cieszy się problematyka projektowania systemów baz danych, wykorzystująca wyniki badań z dziedziny reprezentacji wiedzy, inżynierii oprogramowania, semantyki danych, teorii modelu relacyjnego oraz metod projektowania fizycznych plików danych. Dąży się do opracowania metodologii, które poczynając od zdefiniowania wymagań użytkownika prowadziłyby do implementacji określonego zastosowania, a także do opracowania automatycznych narzędzi wspomagających projektowanie systemu bazy danych.

W końcu roku 1985 znana oficyna wydawnicza North-Holland wydała monografię zatytułowaną „Computer-Aided Database Design: the DATAID Project" (ISBN: 0444877355). Książka ta jest formą podsumowania wyników podjętego w roku 1980 projektu DATAID, w całości ukierunkowanego na rozwój metodologii wspomaganego komputerowo projektowania baz danych. W projekcie uczestniczyło 8 największych uniwersytetów włoskich i instytutów badawczych oraz kilku partnerów przemysłowych; koordynatorem i sponsorem całości był Włoski Krajowy Komitet Badań Naukowych.

Książka liczy 221 stron i zawiera teksty 11 prac. Pierwsza praca, autorstwa redaktorów książki — A. Albano, V. de Antonellis i A. Di Leva pt. „Computer-Aided Database Design: the Dataid Approach" jest rodzajem wprowadzenia. Charakteryzuje ona ogólnie przyjętą w DATAID metodę projektowania scentralizowanych i rozproszonych baz danych, a także opracowane narzędzia wspomagania procesu projektowania.

De Antonellis i Zonta w pracy pt. „A Tool for Modeling Dynamics in Conceptual Design" przedstawiają INCOD-E — narzędzie zapewniające interakcyjne wspomaganie projektowania struktur baz danych i automatyczne utrzymywanie pełnej historii opracowywanych alternatyw projektowych. Batini i in. „GINCOD: A Graphical Tool for Conceptual Design of Data Base Applications" opisują narzędzie graficzne GINCOD (ang. graphics interactive conceptual design of data, translations and events), wykorzystujące rozwinięty model obiektowo-związkowy (ER) Chena i przeznaczone do wspomagania procesu projektowania na poziomie konceptualnym. Dwie kolejne prace (Albano i Orsini — „A Sottware Engineering Approach to Database Design: the Galileo Project" oraz Capaccioli i Occhiuto — „A Workbench for Conceptual Design in Galileo") są poświęcone opisowi konstrukcji i stosowania języka Galileo. Język ten także został opracowany z myślą o wspomaganiu projektowania na poziomie konceptualnym. Umożliwia on definiowanie abstrakcyjnych typów danych, definiowanie klas opisujących abstrakcyjne mechanizmy semantyki modelu danych oraz definiowanie zbioru transakcji wykonywanych na bazie danych.

Duży zespół autorski — Bert i in. — przedstawia bardzo interesującą i ważną pracę, pt. „The Logical Design in the Dataid Project: the Easymap System". Autorzy opisują system ISIDE (ang. integrated system for implementation design) wspomagający zarówno fazę projektowania logicznego, jak i projektowania fizycznego bazy danych. Przedstawiono ogólną architekturę systemu oraz strukturę tzw. metabazy danych, w której są pamiętane wszystkie dane wykorzystywane w procesie projektowania. Przedstawiono także system EASYMAP przeznaczony do translacji specyfikacji konceptualnych na odpowiadające im specyfikacje logiczne, zarówno dla baz danych relacyjnych, jak i baz danych sieciowych typu CODASYL.

Maio, Santori i Scalas w pracy „Architecture of a Physical Design Tool for Relational DBMSs" opisują oprogramowanie IDEA, wspomagające projektowanie fizyczne relacyjnej bazy danych. IDEA (ang. index design algorithm) na podstawie logicznej struktury danych oraz zbioru oczekiwanych transakcji określa zbiór indeksów umożliwiających uzyskiwanie dostępu do danych przy najniższym koszcie całkowitym. IDEA jest dostosowana do systemów zarządzania relacyjnymi bazami danych o architekturze i charakterystyce zbliżonej do Systemu R.

Zagadnieniem projektowania fizycznego, lecz w odniesieniu do baz danych typu CODASYL, zajmują się również Orlando i in. w pracy „Integrated Tools for Physical Database Design in CODASYL Environment". Moduł EOS, korzystając z opisu struktury logicznej i fizycznej, opisu transakcji i charakterystyki sprzętu, szacuje sprawność bazy danych, m.in. mierząc obciążenie systemu i czasy odpowiedzi dla różnych scenariuszy wykonywania transakcji. Z kolei moduł EROS na podstawie wyników produkowanych przez moduł EOS szacuje koszty alternatywnych rozwiązań projektowych i określa optymalny schemat fizyczny oraz odpowiadającą temu schematowi implementację transakcji.

Bardzo wartościowa, dotycząca nowoczesnego kierunku jest praca Ceriego i Perniciego pt. „DATAID-D: Methodology for Distributed Database Design". W pracy omówiono wiele ważnych aspektów projektowania rozproszonych baz danych. Szczególną uwagę zwrócono na pionową i poziomą fragmentację relacji, alokowanie fragmentów relacji oraz rekonstrukcję schematów na stanowiskach lokalnych systemu rozproszonej bazy danych.

Książkę kończy praca S. Navathe pt. „Important Issues in Database Design Methodologies and Tools". Autor ten nie brał wprawdzie bezpośredniego udziału w pracach projektu DATAID, lecz jako wybitny specjalista problematyki baz danych został przez koordynatorów zaproszony do opracowania artykułu stanowiącego swoistego rodzaju recenzję dokonań włoskich zespołów. Trzeba stwierdzić, że ocena ta wypada jednoznacznie pozytywnie.

Wydaje się, że zarówno idea projektu DATAID, jak uzyskane i dokumentowane omawianą książką wyniki są osiągnięciami unikatowymi w skali światowej. Autorom tej recenzji nie jest znany żaden inny, spójny system metod i zrealizowanych narzędzi programowych, który ujmowałby zagadnienia projektowania baz danych w sposób równie szeroki, dogłębny i praktycznie przydatny. Książka jest pozycją bardzo ważną, godną polecenia wszystkim, którzy interesują się zagadnieniami projektowania baz danych.

PIOTR J. JASIŃSKI
ZBYSZKO KRÓLIKOWSKI
JACEK SZULCZYŃSKI

Przypominamy Czytelnikom, że 31 sierpnia upływa termin wnoszenia wpłaty na prenumeratę INFORMATYKI w czwartym kwartale bieżącego roku

Terminologia polskiego Logo (1)

Przy okazji publikowania artykułu nt. składni Logo (str. 10), warto zastanowić się nad propozycjami spolszczenia tego języka, tzn. zastąpienia angielskich słów zastrzeżonych polskimi. Powstałe w ubiegłym roku polskie Logo — znane pod nazwą PTI Logo — jest już faktem. Choć najprawdopodobniej nie dobór terminologii jest największą wadą polskiego interpretera Logo, lecz niektóre defekty odziedziczone po wersji angielskiej, w rubryce terminologicznej wypada zająć się tylko analizą i krytyką polskojęzycznych odpowiedników angielskich nazw instrukcji, odkładając ten ważniejszy temat na inną okazję.

Autorzy polskiej wersji Logo postanowili spolszczyć wszystkie słowa zastrzeżone. Nie jest to jednak proste. Najłatwiej wygląda sprawa z nazwami funkcji trygonometrycznych, które są międzynarodowe. Nie ma tu więc co spolszczać — COS, CTG, SIN, TG, ARCCOS, ARCSIN, ARCCTG, ARCTG mają brzmienie prawie identyczne z angielskim. Jednak w tym miejscu nasuwa mi się pierwsza wątpliwość. Czy można spolszczyć nazwy innych operacji matematycznych? Niektóre z nich są także uznane za międzynarodowe, np.. SQRT, AND, NOT, OR czy wartości logiczne FALSE i TRUE. Są one oznaczeniami pewnych działań, podobnie jak znaki +, —, *, lub — pewnych wartości, jak π, e itp. Nie ma więc sensu ich zmieniać, nawet w dobrej wierze, gdyż może to przynieść więcej szkody niż pożytku. Gdyby więc mój głos liczył się w dyskusji przy ustalaniu tych nazw, to głosowałbym za utrzymaniem nazw oryginalnych w wymienionych wypadkach.

Natomiast spolszczenie nazw niektórych operacji, np. DIV (na ILORAZ), PRODUCT (na ILOCZYN) i SUM (na SUMA) jest bardziej właściwe, gdyż odpowiada nadaniu polskich nazw podprogramom, co jest powszechnie stosowane w programowaniu. Podobna uwaga odnosi się do operacji REMAINDER (zamieniona na RESZTA) i EQUALP (na ROWNE?). Nie uważam jednak za stosowne spolszczenie nazwy funkcji RANDOM w sytuacji, gdy we wszystkich innych językach programowania ma ona przyjętą nazwę (RND lub RAND), tym bardziej że INT (czyli INTEGER) zamieniono w wersji polskiej na ENTIER. Wahałbym się też bardzo, czy spolszczyć nazwę funkcji ROUND, jak to uczyniono, przyjmując nazwę ZAOKRĄG.

Równie ważne, a może nawet ważniejsze od operacji arytmetycznych i logicznych, są w Logo operacje graficzne. Polskim odpowiednikom angielskich nazw instrukcji należy więc przyjrzeć się szczególnie dokładnie. Polskie nazwy najprostszych instrukcji FORWARD i BACK (NAPRZÓD i WSTECZ) można przyjąć bez zastrzeżeń. Jeżeli jednak wskazują one kierunek poruszania się żółwia po ekranie, to należy być konsekwentnym i przyjąć nazwy WLEWO (zamiast LEWO) i WPRAWO (zamiast PRAWO) na oznaczenie instrukcji LEFT i RIGHT. Nazwy innych ważnych operacji, sprawdzania położenia żółwia XCOR i YCOR, zostały dobrane trafnie (XPOZ↑, YPOZ↑), lecz nie można chyba tego powiedzieć o nazwach operacji ustawiania żółwia w zadane położenie SETX i SETY. Choć rozumiem zasadę, która przyświecała autorom polskiego Logo, polegającą na tym, aby instrukcje odczytywania i nadawania wartości miały możliwie zbliżone nazwy, uważam że nie jest to korzystne, gdyż może prowadzić do wielu pomyłek. Tak więc nazwy XPOZ i YPOZ, będące odpowiednikami angielskich SETX i SETY, różnią się zbyt mało od wyżej podanych nazw sprawdzania położenia żółwia. Dlatego sądzę, że celowe byłoby przyjęcie innych nazw, np. POZX i POZY, co jest zgodne z przyjętym odpowiednikiem nazwy SETPOS, oznaczającej ustawienie żółwia przez podanie obu współrzędnych jednocześnie (POZ, choć z wymienionych względów powinno być raczej POZXY). Warto przy tym dodać, że operację sprawdzania położenia żółwia przez równoczesne odczytanie obu współrzędnych, POSITION, nazwano po polsku POZ↑.

Oprócz przesuwania żółwia w zadane położenie ważny jest również kierunek jego poruszania. Do tego celu służą instrukcje:

- sprawdzania kąta poruszania się żółwia, HEADING, którą przetłumaczono, chyba trafnie, na KĄT↑,
- ustawiania żółwia pod zadanym kątem, SETHEADING, przetłumaczone na KĄT,
- sprawdzania nachylenia odcinka łączącego żółwia z danym punktem, TOWARDS, co przetłumaczono na AZYMUT.

Uważam, że w drugim wypadku lepiej byłoby wyraźniej odróżnić instrukcję ustawienia kierunku ruchu żółwia od sprawdzania tego kierunku, mówiąc np. KURS, a nie KĄT. Termin AZYMUT jest natomiast terminem wojskowym lub co najwyżej geograficznym, a nie matematycznym, dlatego sądzę, że lepiej byłoby nazwac odpowiednią instrukcję wyrazem KIERUNEK.

Instrukcje przesłaniania żółwia, HIDETURTLE (po polsku SŻ, schowaj żółwia), odsłaniania żółwia (PŻ, pokaż żółwia) i sprawdzania, czy żółw jest widoczny SHOWNP (WIDAĆ?) też można nazwać po polsku lepiej ze względów semantycznych (np. ZASŁOŃ, ODSŁOŃ, WIDOCZNY?), choć może jest to mniej ważne.

Do operacji graficznych zalicza się także operowanie piórkiem, a nie pisakiem — jak przyjęto w polskim Logo — gdyż ten termin w informatyce oznacza raczej całe urządzenie kreślące, tzn. plotter. Nie widzę powodu, aby operacje opuszczania i podnoszenia piórka, PENDOWN i PENUP, nazywać po polsku skrótami OPU i POD, zamiast pełnymi nazwami OPUŚĆ i PODNIEŚ. Dwie pozostałe operacje dotyczące pisaka, PENERASE i PENREVERSE, mają również postać poleceń, tak więc lepiej byłoby nazwać je ŚCIERAJ i ODWRÓĆ, zamiast ŚCIERANIE i ODWRACANIE.

Ważna grupa operacji graficznych dotyczy operowania kolorem. Sprawdzanie koloru tła, piórka i tekstu, BACKGROUND, PENCOLOUR, TEXTCOLOUR, nazwano w polskim Logo TŁO↑, PISAK↑ i KOLORYT↑. Sądzę, że ze względów terminologicznych lepiej byłoby mówić PIÓRO↑ zamiast PISAK↑ i TEKST↑ zamiast KOLORYT↑. Natomiast zamiast nazw TŁO, RAMKA, PISAK i KOLORYT, na oznaczenie operacji ustawiania koloru tła, marginesu, piórka i tekstu, tzn. SETBG, SETBORDER, SETPC i SETTC, zaproponowałbym wyrazy: KOLTŁA, KOLRAMKI, KOLPIÓRKA i KOLTEKSTU. Jeśli jednak trzeba by przyjąć zasadę, że nazwy instrukcji sprawdzania odpowiednich właściwości (tj. odczytu wartości) nie powinny znacznie różnić się od nazw instrukcji zmiany tych właściwości (tj. nadawania wartości), to na oznaczenie operacji ustawiania kolorów proponowałbym wyrazy TŁO, MARGINES, PIÓRKO i TEKST.

Nazwy innych operacji graficznych, jak CLEAN, CLEARSCREEN, HOME, dotyczących operowania całym obrazem, obrazem i żółwiem, bądź samym żółwiem, nazwane po polsku ZMAŻ, CZYŚĆ i WRÓĆ, pozostawiłbym, proponując zamienić miejscami nazwy ZMAŻ i CZYŚĆ (a poprawniej — OCZYŚĆ, bo jest to czynność jednokrotna), zgodnie z angielskim znaczeniem wyrazu CLEAN (czyścić) i znaczeniem odpowiednich pojęć (CLEAN — oczyszczenie ekranu, CLEARSCREEN — zmazanie wszystkiego), tym bardziej że instrukcję CLEARTEXT nazwano po polsku ZMAŻTEKST.

Angielska nazwa operacji SCRUNCH oznacza spłaszczenie, w tym wypadku — stosunek kroku żółwia w kierunku pionowym do kroku w kierunku poziomym, czyli spłaszczenie przestrzeni ruchu żółwia. Dlatego zamiast polskiej nazwy operacji ustawiania tego stosunku, PROPORCJA, która kojarzy się niepotrzebnie z bliżej nieokreśloną operacją arytmetyczną, zaproponowałbym nazwanie jej SPŁASZCZ, a operację sprawdzania aktualnie obowiązującego stosunku nazwałbym SPŁASZCZENIE zamiast PROPORCJA↑.

Choć nazwy operacji graficznych nie są znormalizowane tak jak nazwy większości operacji arytmetycznych lub logicznych, niektóre z nich są używane tak często, że można

je uznać za ustalone. Przedyskutowane propozycje wielu tych nazw przedstawiono w Informatyce nr 2, 3, 4 i 8 z 1985 roku. Tak więc, instrukcja WINDOW, która umożliwia żółwiowi poruszanie się po całej dostępnej przestrzeni, czyniąc ekran jedynie jej prostokątnym wycinkiem, powinna nazywać się po polsku OKNO. Tak też postąpili autorzy polskiego Logo. Instrukcja WRAP powoduje przechodzenie żółwia na przeciwległy brzeg ekranu za każdym razem, gdy w czasie ruchu przekroczy on obszar ograniczony ekranem z drugiej strony. Dlatego tej instrukcji nadałbym polską nazwę ZAWIJANIE (zgodnie z sugestią zawartą w Informatyce nr 4/1985), a nie SKLEJ. Nazwanie operacji FENCE, polegającej na ograniczeniu dopuszczalnego obszaru ruchu dla żółwia, polskim terminem POLE, uważam za błąd, ponieważ termin ten nie wyraża rzeczywistego sensu operacji, sugerując raczej obliczanie pola powierzchni. Ponieważ w operacji FENCE chodzi o ograniczenie obszaru poruszania się żółwia wyłącznie do samego ekranu, nazwałbym ją OGRODZENIE, zgodnie ze znaczeniem angielskiego wyrazu fence (płot).

Inne nazwy przywodzące na myśl operacje graficzne, BRIGHT i FLASH, dotyczą w zasadzie operacji tekstowych i oznaczają rozjaśnienie i migotanie TEKSTU, co chyba trafnie określono nadając im polskie nazwy JASKRAWO i MIGAJ. O ile operacja ustalania współrzędnych aktualnego położenia kursora w tekście, CURSOR, została nazwana po polsku rozsądnie, KURSOR↑, to mam wątpliwości, wynikające z przyczyn przedstawionych powyżej, czy ustawianie kursora w zadane położenie, SETCUR, powinno nazywać się tak bardzo podobnie, tj. KURSOR (może lepiej byłoby RUSZKURSOR). Nie mam natomiast żadnych wątpliwości co do nazw POZYTYW i NEGATYW na oznaczenie poleceń przyjęcia trybu wyświetlania jasnego tekstu na ciemnym tle i ciemnego na jasnym tle, które w wersji angielskiej nazwano NORMAL i INVERSE. Instrukcję przejścia do pracy w trybie tekstowym, TEXTSCREEN, proponowałbym nazwać po polsku TEKSTOWO, zamiast TEKSTY, w celu lepszego odróżnienia od proponowanej nazwy operacji sprawdzania koloru tekstu.

JANUSZ ZALEWSKI

Listy

Warszawa, 2 listopada 1986

Szanowna Redakcjo

Na wstępie pragnąłbym pogratulować szybkości: INFORMATYKA nr 4, 1986, ukazująca się na początku września br., prezentuje artykuł z BYTE'A nr 9'1986! Ale już poważniej — to oczywiście błąd techniczny, a artykuł M. Garetza „Evolution of the Microprocessor — An informal history" pochodzi z roku 1985.

Mark Garetz, w swoim czasie projektant systemu CompuPro 8085/8088, obecnie związany z firmą Viasyn Co., uznał za stosowne opatrzyć artykuł stwierdzeniem, iż przedstawiając swoją historię mikroprocesora przeprasza, jeśli jego wersja nie pokrywa się z poglądami czytelników. „Wyrażone w tym artykule opinie są moimi i mogą, ale nie muszą opisywać rzeczywistości dostrzeganej przez innych" — tak brzmią w dosłownym tłumaczeniu słowa autora.

Artykuł (ten z BYTE'A) należałoby uznać za pożyteczny, gdyby nie tendencyjność autora i pewne usterki tłumaczenia na język polski. Upraszczając nieco problem, wydaje się, że Mark Garetz najchętniej widziałby istnienie tylko jednej, ogólnoświatowej rodziny mikroprocesorów, wywodzącej się żywcem z 8080, co zresztą można zrozumieć, zważywszy na jego osobiste zainteresowania i dokonania.

Pominięcie lub zniekształcenie pewnych sformułowań w efekcie zmienia sens artykułu, co dla wyczulonego na wszelką krytykę dokonań firmy Zilog ucha (mojego) brzmi nieco fałszywie. Najpierw oddajmy głos autorowi. Pisze on m.in., że przewaga, jaką miał mikroprocesor 8085 nad 8080, nigdzie nie była zbliżona do osiągnięć Z80, a nie — cytuję tłumaczenie: „Nowy mikroprocesor miał kilka dodatkowych rozkazów, ale ogólnie udoskonalenia wprowadzone przez Intela były podobne do rozwiązań Ziloga [...]. Firma zadecydowała, że nowy 16-bitowy procesor ma być prostym rozwinięciem 8080". Następnie pominięto zdanie: „Niestety projektanci nie zachowali bezpośredniej kompatybilności z kodem 8080". Dalszy fragment o odpowiedniości rejestrów 8080-8086 został przetłumaczony. Dalej, przy omawianiu MC68000: „Jeśli jakiś rozkaz nie działa prawidłowo, można ustalić (a nie „umieścić"; jaki jest sens umieszczania błędnego rozkazu? — przyp. M.Ch.) go w pamięci ROM i, z ograniczeniami, nawet zmienić listę instrukcji mikroprocesora, jeśli tego potrzebujemy". Jeszcze jeden drobiazg — istnieje pewna subtelna różnica w znaczeniu zwrotów „is called" i „that I call", której niestety nie widać w tłumaczeniu zwrotu „[Z8000] ... that I call the kitchen-sink processor" na „złośliwie nazywany" zamiast „który ja nazywam". Wydaje mi się, że uogólnianie prywatnej nazwy autora na wyrób konkurencji nie było konieczne, zwłaszcza wobec przytoczonej na wstępie uwagi autora o prywatności opinii przedstawianych w artykule.

A skoro już mowa o prywatnych opiniach — moja opinia mówi, że procesory Ziloga nie są aż tak złe, jak to od czasu do czasu można wyczytać w INFORMATYCE, a poza tym — wypada być konsekwentnym nie tylko przy omawianiu tych „innych, gorszych" (niż 8080 i 8086, rzecz jasna). A to, iż Intel żyje przede wszystkim dzięki firmie IBM — to widać także i w Polsce. Zilog tak dużych pieniędzy nie ma, a z tego, co posiada, zrobił już i tak dużo dobrych układów. [...]

MICHAŁ CHOROSZUCHA
Warszawa

Szanowny Panie

Pana list przyczytałem z przyjemnością, gdyż świadczy on o żywym zainteresowaniu treścią INFORMATYKI. Jeśli temu zainteresowaniu towarzyszy krytyczne spojrzenie na zawartość czasopisma, to wypada mi się tylko cieszyć, tym bardziej że robi to tak uważny Czytelnik i ceniony Autor.

Odpowiadając, wypada przyznać się do części zauważonych przez Pana błędów, lecz nie do wszystkich. Wspomniany artykuł zamieszczony w INFORMATYCE jest zaledwie opracowaniem a nie tłumaczeniem, co zapewnia nam pewien margines działania, dotyczący m.in. wierności tłumaczenia. Proszę więc wybaczyć, że nie sprostaliśmy Pana wymaganiom pod tym względem, ale nie czynimy tego z zasady, chyba że wątpliwości dotyczą nie tyle niuansów interpretacyjnych, co treści merytorycznej.

Oddzielną sprawą jest kwestia wiedzy i umiejętności autorów, tłumaczy i innych osób współpracujących z INFORMATYKĄ. Praca tłumacza jest dla mnie, ponad wszelką wątpliwość, działalnością twórczą i trudno oczekiwać, by dobrzy tłumacze rodzili się na kamieniu. INFORMATYKA płaci znacznie mniej niż warte jest na rynku dobre tłumaczenie lub opracowanie, w związku z tym trudno nam nakłonić do współpracy najlepszych. Próbujemy więc wychować sobie zespół, angażując do pracy początkujących tłumaczy i starając się pomagać im w doskonaleniu warsztatu. Oczywiście, niewielu z nich osiąga wysoki poziom, a nawet ci często odchodzą, ulegając różnym przeciwnościom. Jeżeli zechce Pan przekonać się o tym, jak trudna to praca, jako autor lub tłumacz przyszłej książki, to z przyjemnością ją Panu zrecenzuję.

Przykro mi, że z powodu braku miejsca nie możemy wydrukować Pana ciekawego listu w całości. Natomiast propozycję napisania artykułu nt. procesorów Z80000 oraz iAPX 386 z chęcią przyjmę, jeśli będzie on rozwinięciem wydrukowanego już artykułu prof. Daniela Tabaka. Po szczegółowe materiały oryginalne dotyczące obu mikroprocesorów proszę skontaktować się z redakcją.

JANUSZ ZALEWSKI
Z-ca redaktora naczelnego

Kowalski R.: Logic in expert systems (1)
INFORMATYKA 1987, No. 5, p. 1

Characteristics of logic programming and relationship between expert systems and rule systems.

Kowalski R.: Logik in Expertensystemen (1)
INFORMATYKA 1987, Nr. 5, S. 1

Eine Charakteristik der logischen Programmierung und der Zusammenhänge zwischen Expertensystemen und Regelsystemen.

Kowalski R.: Logika w systemach ekspertowych (1)
INFORMATYKA 1987, nr 5, s. 1

Charakterystyka programowania logicznego oraz związków pomiędzy systemami ekspertowymi a systemami regułowymi.

Węglarz J.: Evaluation of computer system operation — object, methods, development directions (2)
INFORMATYKA 1987, No. 5, p. 4

Second part of the paper, which includes discussion of load modelling and some development directions of computer system operation evaluating methods.

Węglarz J.: Beurteilung der Wirkung von Computersystemen — Objekt, Methoden und Entwicklungsrichtungen (2)
INFORMATYKA 1987, Nr. 5, S. 4

Zweiter Teil eines Artikels, der eine Besprechung der Belastungsmodellierung, sowie mancher Entwicklungsrichtungen von Beurteilungsmethoden der Computersystemwirkung umfasst.

Węglarz J.: Ocena działania systemów komputerowych — przedmiot, metody, kierunki rozwoju (2)
INFORMATYKA 1987, nr 5, s. 4

Druga część artykułu, zawierająca omówienie modelowania obciążeń oraz niektórych kierunków rozwoju metod oceny działania systemów komputerowych.

Morzy T.: Transaction synchronization in distributed database systems (1)
INFORMATYKA 1987, No. 5, p. 7

First part of presentation of the method for transaction synchronization in distributed database systems.

Morzy T.: Transaktionsynchronisation in verteilten Datenbanksystemen (1)
INFORMATYKA 1987, Nr. 5, S. 7

Erster Teil einer Besprechung der Methode für Transaktionsynchronisation in verteilten Datenbanksystemen.

Morzy T.: Synchronizacja transakcji w systemach rozproszonych baz danych (1)
INFORMATYKA 1987, nr 5, s. 7

Pierwsza część omówienia metod synchronizacji transakcji w systemach rozproszonych baz danych.

Gurbiel E., Krupicka H., Płoski Z.: Logo syntax — a formalization attempt (2)
INFORMATYKA 1987, No. 5, p. 10

Second part of Logo syntax formalization proposal, which includes discussion of expressions, primary functions and object syntax.

Gurbiel E., Krupicka H., Płoski Z.: Logo-Syntax — eine Formalisationprobe (2)
INFORMATYKA 1987, Nr. 5, S. 10

Zweiter Teil eines Vorschlages für Formalisation von Logo-Syntax, der eine Besprechung von Ausdrücken-, primären Funktionen- und Objektensyntax umfasst.

Gurbiel E., Krupicka H., Płoski Z.: Składnia języka Logo — próba formalizacji (2)
INFORMATYKA 1987, nr 5, s. 10

Druga część propozycji formalizacji składni języka Logo, zawierająca omówienie składni wyrażeń, funkcji pierwotnych i obiektów.

Pawłowski M.: Microprogramming (2)
INFORMATYKA 1987, No. 5, p. 12

Second part of the paper, which contains discussion of problems connected with designing of control circuit for microprogrammed functional set.

Pawłowski M.: Mikroprogrammierung (2)
INFORMATYKA 1987, Nr. 5, S. 12

Zweiter Teil eines Artikels, der eine Besprechung von Projektierungsproblemen einer Steuerschaltung für mikroprogrammierte Funktionsgruppe umfasst.

Pawłowski M.: Mikroprogramowanie (2)
INFORMATYKA 1987, nr 5, s. 12

Druga część artykułu, zawierająca omówienie problemów projektowania układu sterującego dla mikroprogramowanego zespołu funkcjonalnego.

Zalewski J.: Concurrent DOS (1)
INFORMATYKA 1987, No. 5, p. 18

First part of characteristics of the Concurrent DOS operating system.

Zalewski J.: Concurrent DOS (1)
INFORMATYKA 1987, Nr. 5, S. 18

Erster Teil einer Charakteristik von Concurrent DOS-Betriebsystem.

Zalewski J.: Concurrent DOS (1)
INFORMATYKA 1987, nr 5, s. 18

Pierwsza część charakterystyki systemu operacyjnego Concurrent DOS.

WARUNKI PRENUMERATY NA 1987 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu".

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.

Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:

— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

Uwaga!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie ul. Mazowiecka 12, tel. 27-43-65 lub zamówić w Dziale Handlowym Wydawnictwa, ul. Bartycka 20, skr. poczt. 1004, 00-950 Warszawa, tel. 40-37-31, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 150 zł za numer (50 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
450	150	900	300	1800	600

KONIEC TWOICH PROBLEMÓW

Nareszcie wszystkie potrzebne Ci dane dotrą na czas
w przejrzystej formie tabel i wykresów.

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed
Twoim przedsiębiorstwem.

Wyobraź sobie

MILIARD

rekordów, które możesz
zapełnić według własnych potrzeb i uznania.

Miliard kooperantów, miliard pracowników, miliard produktów
— z tym wszystkim nasz MEGA-BANK poradzi sobie bez trudu.

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!

COMPUTER STUDIO KAJKOWSCY
PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW
ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

EO/74/K/87