5
1990

informatyka

Architektury zredukowane
Maszyny lispowe
Protokoły komunikacyjne

WYDAWNICTWO NOT SIGMA

Cena 5800 zł

informatyka

Miesięcznik

ISSN 0542-9951. INDEX 36124

nr 5 1990
Maj
Rok wydania XXV

W numerze:

W najbliższych numerach:

- Jerzy Dawidowski charakteryzuje ręczne rejestratory danych zasilane bateryjnie, przeznaczone do pracy w zmiennym środowisku klimatycznym.
- Zbyszko Królikowski poświęca swój artykuł problematyce optymalizacji przetwarzania zapytań w systemach relacyjnych baz danych.
- Mieczysław Kłopotek zajmuje się rozpoznawaniem słów w mowie ciągłej, co jest istotnym aspektem automatycznego rozumienia mowy.
- Piotr Orwaldi opisuje metody programowania reprogramowalnych pamięci EPROM.
- Mariusz Kuc prezentuje kompilator AdaVantage dla mikrokomputera MacIntosh.

WYDAWCA: Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4,
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja: 01-552 Warszawa,
Pl. Inwalidów 10 p. 128, 136
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

DRUK:
PRASOWE ZAKŁADY GRAFICZNE
RSW „PRASA-KSIĄŻKA-RUCH"
ul. Dworcowa 13,
85-950 BYDGOSZCZ
zam. 981/90
Obj. 4,0 ark. druk. Nakład 5300 egz.

Cena egzemplarza 5800 zł

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji

WARUNKI PRENUMERATY

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia). Blankiety te będą dostarczane dotychczasowym prenumeratorom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu, w Radach Wojewódzkich NOT bądź w Redakcjach czasopism.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: **Państwowy Bank Kredytowy III/O Warszawa nr 370015–7490–139–11.**

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 ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

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.

Wpłaty na prenumeratę przyjmowane są w terminach:

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

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00-716 Warszawa), skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 26-80-16), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00-950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena egzemplarza: normalna 5800 zł, ulgowa 1160 zł

Uwaga! Podane ceny mają charakter wstępny i mogą ulec zmianie, w związku z powyższym Wydawnictwo zastrzega sobie prawo żądania dopłat do już opłaconej prenumeraty.

LESŁAW SIENIAWSKI
Instytut Komputerowych Systemów Automatyki i Pomiarów
Wrocław

Architektury zredukowane

Jednym z zasadniczych wymagań stawianych systemom komputerowym jest – oprócz funkcjonalności i niezawodności – szybkość przetwarzania odpowiednia do potrzeb. Innym jest zmniejszenie tzw. luki semantycznej (konceptualnej), wynikającej ze znacznego zróżnicowania między pojęciami opisującymi fizyczny sprzęt liczący a pojęciami używanymi przez programistów tworzących złożone systemy oprogramowania. Luka ta powstaje więc na skutek konieczności posługiwania się przez twórców oprogramowania obiektami abstrakcyjnymi (np. rekordy, kolejki, listy) i operacjami wykonywanymi na tych obiektach, które nie mają bezpośredniego odpowiednika w architekturze logicznej komputera.

Jedną ze współczesnych tendencji architektonicznych wychodzących naprzeciw wymienionym wyżej postulatom jest koncepcja komputera RISC.

GENEZA RISC

Termin RISC (ang. *Reduced Instruction Set Computer*) pierwsze publikacje (1980) i opracowanie procesora o takich właściwościach są zasługą prof. D. Pattersona, z Uniwersytetu Berkeley w Kalifornii. Po grupie Berkeley, do nowego nurtu badań włączyła się grupa prof. J. Hennesy'ego, z Uniwersytetu Stanford. W 1982 ogłoszono wyniki prac prowadzonych w IBM pod kryptonimem „801", rozpoczętych w 1975 roku, tj. przed D. Pattersonem. Podstawowe idee, nie wiązane jeszcze z terminem RISC, powstały jednak znacznie wcześniej; poniżej omówiono niektóre z uzyskanych wyników.

System IBM/360 wprowadzony w 1964 roku zapoczątkował nowoczesną architekturę komputerów [9]. Wprowadzono wówczas rozróżnienie dwu pojęć: architektury logicznej, tj. architektury widzianej z poziomu rozkazów, oraz jej sprzętowej realizacji; rodzina 360 zawierała różne modele o zróżnicowanej wydajności i cenie, lecz o tych samych właściwościach na poziomie języka maszynowego. Procesor IBM/360 model 44 realizował sprzętowo tylko część architektury; pozostała część była implementowana przez oprogramowanie. Dlatego miał on w swoim czasie istotnie lepszy stosunek kosztu do wydajności. Podobnego eksperymentu dokonano również w systemie 370, przenosząc doń z doświadczalnej maszyny IBM 801 typu RISC głęboko optymalizujący kompilator języka PL/8. Optymalizator traktował IBM/370 jak maszynę o modelu działania rejestr-rejestr. Na poziomie ograniczonej w ten sposób listy rozkazów programy pracowały o 50% szybciej niż dla najlepszych optymalizujących kompilatorów używających pełnego zbioru rozkazów.

VAX – pełna implementacja architektury VAX-11 wymagała użycia 9 układów VLSI (tzw. VLSI VAX). Badania wykazały, że 20% rozkazów komputera wymaga zaangażowania 60% mikrokodu, lecz rozkazy te stanowią tylko 0,2% ogółu wywołań. W komputerze MicroVAX32 rozkazy te zaimplementowano programowo, co pozwoliło na zamknięcie procesora w jednym układzie, z możliwością dołączenia zewnętrznego koprocesora zmiennoprzecinkowego, ograniczenie wykorzystanych zasobów sprzętowych i przyspieszenie działania o 20% (tabela 1).

Kompilator Moduli-2. M. Powell z firmy DEC ulepszył kompilator języka Modula-2 przez okrojenie zbioru możliwych trybów adresowania i okrojenie samego zbioru rozkazów, uzyskując 20% zwiększenie szybkości.

Pierwszą technologiczną innowacją było mikroprogramowanie, zapewniające obszerny zbiór rozkazów za pomocą prostego sprzętu. Wraz z rozwojem pamięci półprzewodnikowych stało się możliwe dalsze wzbogacanie zbioru rozkazów. Przemawiały za tym względy:

– bogatsze zbiory rozkazów umożliwiłyby uproszczenie kompilatorów,
– bogatsze zbiory rozkazów mogłyby przełamać tzw. kryzys software'owy polegający na zwiększeniu kosztów oprogramowania przy postępującym spadku cen sprzętu i pozwolić na przeniesienie do sprzętu maksymalnie wielu funkcji; przez wprowadzenie rozkazów zastępujących instrukcje języków programowania można by zlikwidować lukę semantyczną,
– bogatszy zestaw rozkazów mógłby wreszcie poprawić jakość architektury ocenianą w kategoriach: wielkości programu, liczby bitów przypadających na rozkaz, liczby bitów danych sprowadzanych z pamięci podczas wykonywania rozkazu.

Tabela 1. Efekty redukcji architektury VAX [7]

Parametr	VLSI VAX	MicroVAX32
Liczba US łącznie z FPU	9	2
Objętość mikrokodu	480 Kb	64 Kb
Liczba tranzystorów	1250 tys.	101 tys.
Technologia	NMOS 3,0 μm	NMOS 3,0 μm

W latach siedemdziesiątych wyznawano następujące zasady projektowania maszyn:
– obszerne mikroprogramy nie zwiększają kosztu sprzętu,
– ponieważ mikrorozkazy są szybsze niż rozkazy, więc przesunięcie funkcji oprogramowania do mikrokodu daje korzyści w postaci przyspieszenia obliczeń i zwiększenia niezawodności działania,
– ponieważ czas działania jest proporcjonalny do wielkości programu, techniki architektoniczne prowadzące do małych programów prowadzą również do szybszych komputerów,
– rejestry są staromodne i utrudniają konstruowanie kompilatorów; architektury stosowe i architektury o modelu obliczeń typu pamięć-pamięć są najlepszymi środowiskami do wykonywania programów.
Według tych zasad zbudowano m.in. systemy IBM 370/168, DEC VAX-11/780, Xerox Dorado, Intel iAPX 432 (tabela 2).

Tabela 2. Przykładowe maszyny CISC*) [9]

	IBM 370/168	VAX-11/780	Dorado	iAPX 432
Rok wprowadzenia	1973	1978	1978	1982
Liczba rozkazów	208	303	270	222
Objętość PAS	420 Kb	480 Kb	136 Kb	64 Kb
Szerokość rozkazu	16–48 b	16–456 b	8–24 b	6–321 b
Technologia	ECL MSI	TTL MSI	ECL MSI	NMOS
Model wykonywania obliczeń	R-M M-M R-R	R-M M-M R-R	S	S M-M
Pojemność PAP	64 Kb	64 Kb	64 Kb	0

Oznaczenia: R – rejestr, M – pamięć, S – stos
*) CISC (ang. *Complex Instruction Set Computer*) – komputer o złożonej liście rozkazów

Wyeliminowanie drogiej pamięci rdzeniowej i gwałtowny rozwój technologii półprzewodników podważyły jednak wypracowane zasady:
– pamięć półprzewodnikowa, która zastąpiła rdzeniową, zmieniła relacje między czasem dostępu do pamięci sterującej a czasem dostępu do pamięci operacyjnej; ta ostatnia przestała być 10-krotnie wolniejsza od pamięci mikroprogramów,

– nakłady na opracowanie mikrokodu okazały się znaczące z powodu trudności w usuwaniu błędów; charakter tej pamięci zmienił się stopniowo z ROM na RAM,
– twórcy prostych kompilatorów nie umieli wykorzystywać nowych

funkcji wprowadzonych dla usunięcia luki semantycznej. Nie spełniły się też oczekiwania architektów co do praktycznego wykorzystania możliwości pisania programów na poziomie mikrokodu, z uwagi na złożoność technologii mikroprogramowania. Wprowadzenie wirtualnej organizacji pamięci (IBM 360/67) oraz wielozadaniowości potencjalnie wymuszającej przeładowywanie PAS przy każdym uaktywnianiu nowego procesu w systemie, dodatkowo tę trudność powiększyło.

Z drugiej strony, wprowadzenie pamięci pomocniczej pozwoliło na dostęp do PAO równie efektywny jak do PAS; przewaga mikroprogramowej realizacji funkcji nad programową przestała istnieć. Pozostało jedynie aktualne wymaganie, by proste operacje programowe były wykonywane w jednym cyklu.

Nowa „filozofia" sformułowana w 1980 r. przewidywała następujące zasady projektowe;
– funkcje rozkazów powinny być proste, chyba że istnieje uzasadniony powód, aby było inaczej; zwiększenie cyklu maszyny spowodowane wprowadzeniem złożonej funkcji rozkazowej musi być zrekompensowane odpowiednim zmniejszeniem liczby cykli na rozkaz,
– mikrorozkazy nie powinny być szybsze od pozostałych rozkazów,
– przesuwanie funkcji do mikrokodu nie ulepsza ich, ale czyni je trudniejszymi do zmian,
– proste dekodowanie i potokowa realizacja rozkazów są ważniejsze niż wielkość programu,
– zamiast generowania złożonych rozkazów powinna być użyta odpowiednia technika kompilacji; kompilatory dla komputerów RISC winny minimalizować złożoność obliczeń na etapie przygotowania kodu wynikowego i optymalizować użycie rejestrów, zamiast poszukiwać idealnego trybu adresowania i najkrótszego formatu rozkazu.

Przegląd implementacji

W 1987 r. już ponad 10 producentów oferowało komercyjne produkty typu RISC. W tabeli 3 przedstawiono główne cechy czterech doświadczalnych i dziesięciu przemysłowych wyrobów tej klasy.

Tabela 3. Przegląd konstrukcji RISC [14]

System	Rozkazy			Tryby adreso - wania	Rejestry uniwer- salne	Stero- wanie ukła- dowe
	liczba	formaty	1- cy- klowa			
RISC II	39	2	T	2	138	T
MIPS-X	31	4	T	2	32	T
IBM 801	120	2	T	3	32	T
MIRIS	64	4	T	2	2048	N
Pyramid	128	3	N	16	528	N
Ridge	128	3	N	2	16	N
Acorn RISC	44	6	T	2	16	T
T800	111	1	T	1	6	N
IBM RT PC	118	2	N	2	16	N
HP 9000	140	1	T	2	32	T
MF 1600	70	0	N	0	1024	T
Clipper	101	14	T	9	16	T
MIPS R2000	102	3	T	4	32	T
Am 29000	115	1	T	1	192	T

Oznaczenia: T – tak, N – nie

IBM 801

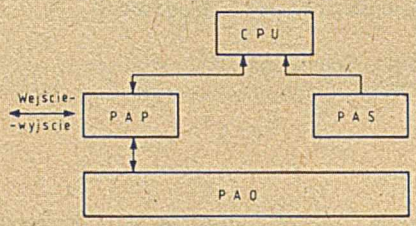
Celem badawczym projektu IBM 801 [4,9,13] było sprawdzenie współczesnych trendów w kierunku złożoności architektury. System ten miał być systemem sprzętowo-programowym, w którym programowanie prawie w całości miało być wykonywane w języku wysokiego poziomu; poszukiwano odpowiedzi na pytanie, w jaki sposób można tanio zwiększyć moc obliczeniową w celu zwiększenia wydajności programisty. Wyniki eksperymentów wykazały, że większość spośród ponad 200 rozkazów maszyny jest wykonywanych bardzo rzadko, a rozkład częstości użycia rozkazów jest zdeterminowany przez rozkazy typu *LOAD*, *STORE*, *BRANCH* i kilka prostych operacji na rejestrach (tabela 4).

Jak widać, dziesięć rozkazów pokrywało łącznie 2/3 wystąpień wszystkich rozkazów. M. Hopkins [4] podał, że rozkazy *LM* i *STM* stanowiły łącznie 2,4% wywołań, zaś rozkazy *M* i *D* odpowiednio 0,115% i 0,111% wykonań. Obliczenia naukowe i przetwarzanie danych nieznacznie tylko wpływały na charakter rozkładu.

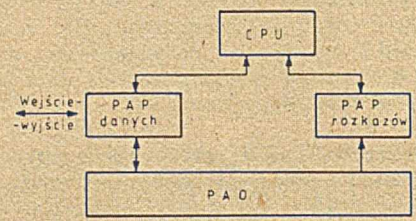
Tabela 4. Rozkład częstości wykonywania rozkazów w modelu IBM 360/85 [4]

Kod rozkazu	Nazwa rozkazu	Częstość użycia [%]
BC	Branch	20,16
L	Load Word	15,49
TM	Test under Mask	6,06
ST	Store Word	5,88
LR	Load Register to Register	4,70
LA	Load Effective Address	4,04
LTR	Load and Test Register	3,78
BCR	Branch on Register	2,69
MVC	Move Characters	2,10
LH	Load Halfword	1,88
*	pozostałe rozkazy	33,22
	RAZEM	100,00

Ogólna struktura IBM 801 różniła się znacznie od przyjętej dla serii 370 (rys. 1 i 2).

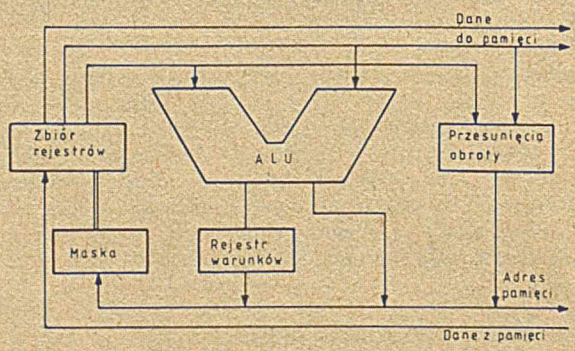


Rys. 1. Struktura IBM 370 [5]



Rys. 2. Struktura IBM 801 [5]

Wbudowane rozkazy były realizowane w jednym cyklu. PAS została jakby zamieniona na PAP rozkazów, nastąpiło rozdzielenie szyny danych od szyny adresów, przez co potencjalnie podwoiła się przepustowość dostępu do PAO. Doprowadzenie wejścia-wyjścia bezpośrednio do pamięci ograniczyło zjawisko zatykania PAP transmisjami. Na rysunku 3 przedstawiono uproszczony schemat przepływu danych w procesorze.



Rys. 3. Przepływ danych w procesorze IBM 801 [5]

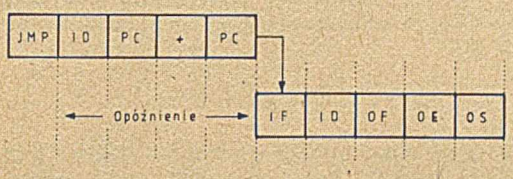
Dzięki 1-cyklowej realizacji rozkazów uzyskano łatwość obsługi przerwań – możliwe było badanie warunku wystąpienia przerwania i jego obsługa wyłącznie między rozkazami, co dodatkowo uprościło strukturę sterowania, z powodu braku konieczności zapamiętywania stanu w chwili przerwania.

W celu skrócenia czasu bezczynności CPU związanego z realizacją skoków w warunkach przetwarzania potokowego, wprowadzono dwa rodzaje rozkazów skoku:

- rozkaz „skocz i wykonaj" (ang. *Branch and Execute*),
- „zwykły" skok (ang. *Regular Branch*).

Pierwszy z nich eliminował możliwość wystąpienia hazardu w trakcie potokowej realizacji rozkazów, przez użycie metody opóźnionego skoku (ang. *Delayed Branch*). Ponieważ adres następnego rozkazu do wykonania jest ustalony dopiero w ostatniej fazie realizacji skoku efektywnego, należy na chwilę zatrzymać przesuwanie rozkazów między kolejnymi stopniami potoku, aby odczekać na pojawienie się wskazanego przez skok rozkazu do wykonania (rys. 4). Generator kodu maszynowego w kompilatorze wstawia po każdym rozkazie skoku jeden lub więcej rozkazów pustych (NOP). Z kolei optymalizator kodu usiłuje przesunąć lub powtórzyć bardziej użyteczne rozkazy, tak aby zajęły one miejsce rozkazów NOP. Firma IBM podała, że przeciętnie udawało się to wykonać w 60% sytuacji. Podczas realizacji programu rozkaz umieszczony bezpośrednio po rozkazie skoku jest wykonywany niezależnie od tego, czy skok jest efektywny czy nie.

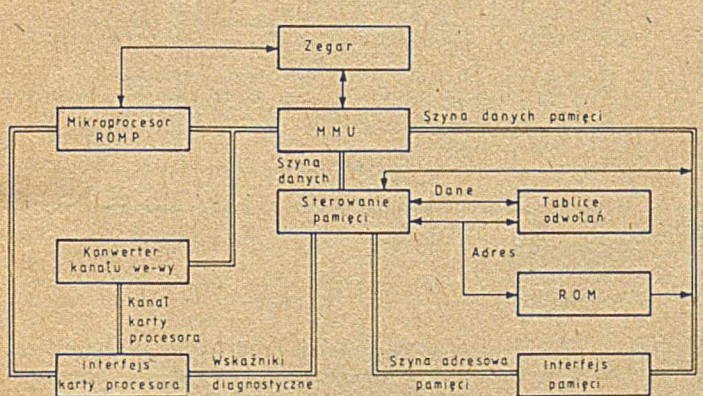
Drugi rodzaj skoku był stosowany wówczas, gdy nie można było znaleźć rozkazu, który można umieścić po skoku. Podczas sprowadzania rozkazu wskazanego przez zwykły skok, CPU blokuje przesuwanie rozkazów między stopniami potoku, w sposób przyjęty w konwencjonalnych realizacjach maszyn z przetwarzaniem potokowym. W odróżnieniu od innych komputerów typu RISC, blokada ta wykonywana jest głównie środkami sprzętowymi.



Rys. 4. Opóźnienie w potoku dla rozkazu skokowego [2]
Oznaczenia:
JMP – kod rozkazu skoku,
ID – dekodowanie rozkazu, OF – pobranie argumentu,
PC – dostęp do licznika rozkazów, OE – przetworzenie argumentu,
IF – kod rozkazu badania warunków, OS – zapamiętanie wyniku

IBM 801 został zrealizowany w technologii ECL (mały i średni stopień scalenia) z cyklem maszynowym 66 ns i osiągał znaczne szybkości działania. Przykładowo, wewnętrzna pętla sortowania stogowego była wykonywana w 35 cyklach, podczas gdy to samo zadanie zajmowało 96 dłuższych cykli IBM 370/168. Wielkość kodu maszynowego stanowiła 0,9 wielkości analogicznego kodu 370/168.

Przemysłowym potomkiem IBM 801 jest ROMP (ang. *Research /Office product division Micro Processor*). ROMP realizuje 118 rozkazów, ma szybką 32-bitową szynę pamięci, przez którą współpracuje ze standardowymi układami pamięciowymi DRAM o czasie dostępu 150 ns i pojemności 256 Kb. W jednym cyklu maszynowym trwającym 170 ns można przesłać adres i dane. Korzystając z procesora ROMP zbudowano płytę komputera IBM RT PC [5,10] (rys. 5). Ma on dużą przestrzeń adresową – 2 TB, zorganizowaną w 4 K segmentów po 256 MB.



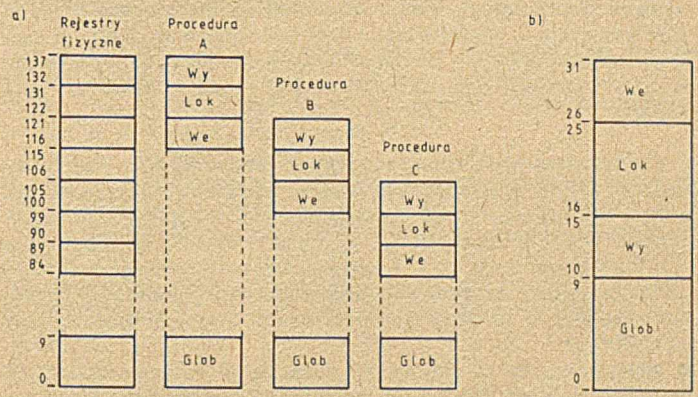
Rys. 5. Schemat blokowy pakietu procesora IBM RT PC [6]

RISC I, RISC II

Prace badawcze zespołu prof. D. Pattersona z Berkeley [3, 9, 13, 14] opierały się na uzyskanych rozkładach mieszanek rozkazowych dla programów nienumerycznych napisanych w językach C i Pascal. Opracowany w 1982 roku 32-bitowy procesor RISC realizował 31 rozkazów. Jego ulepszona wersja z 1983 roku – RISC II – realizowała 39 rozkazów.

RISC II efektywnie wspomagał sprzętowo dwa mechanizmy istotne dla implementacji języków wysokiego poziomu:

- wywoływanie i powrót z podprogramów (ang. *Call/Return*),
- mechanizm opóźnionego skoku.

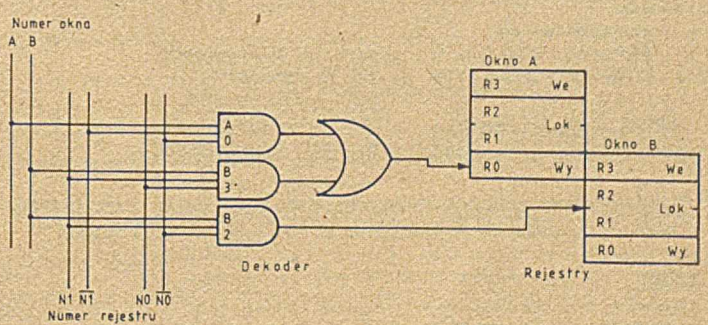


Rys. 6. Okna rejestrowe RISC II
a) dostępność rejestrów fizycznych dla procedur [1],
b) struktura wirtualnego okna rejestrów [3]
Oznaczenia:
Wy – rejestry wyjściowe, We – rejestry wejściowe,
Lok – rejestry lokalne, Glob – rejestry globalne

Do realizacji wywołania podprogramów wprowadzono okna rejestrowe (ang. *overlapped register windows*) – zespół 138 rejestrów fizycznych zawierający 10 rejestrów globalnych dostępnych dla każdego podprogramu oraz 128 rejestrów, z których aktywnemu podprogramowi udostępnia się 32 (rys. 6a). Na rysunku 6b przedstawiono strukturę wirtualnego zbioru (okna) rejestrów dostępnych podprogramowi. Zbiór ten zawiera:

- 10 rejestrów globalnych,
- 6 rejestrów wejściowych, w których podprogram wywołujący umieszcza parametry wejściowe dla danego podprogramu,
- 10 rejestrów lokalnych, dla wewnętrznego wykorzystania przez podprogram,
- 6 rejestrów wyjściowych, w których dany podprogram umieszcza parametry wyjściowe dla podprogramu wywoływanego przez siebie.

Prosty mechanizm sprzętowy zapewnia kojarzenie właściwych rejestrów fizycznych z aktywnym oknem (rys. 7)



Rys. 7. Dekodowanie rejestrów dla dwu okien (RISC) [8]

W celu zapewnienia dowolnej głębokości wywołań podprogramów, zbiór rejestrów fizycznych zorganizowano w tablicę cykliczną, z którą są związane dwa wskaźniki przechowywane w PSW: wskaźnik okna bieżącego (CWP) i wskaźnik najmłodszego okna (SWP). Kiedy głębokość wywołań podprogramów przewyższa pojemność zbioru rejestrów (CWP próbuje osiągnąć wartość SWP), występuje przerwanie i jest uruchamiany podprogram zachowujący rejestry w pamięci. Stwierdzono, że najlepszą strategią jest usuwanie tylko jednego okna; stąd zbiór rejestrów o pojemności n okien przechowuje maksimum n-1 okien.

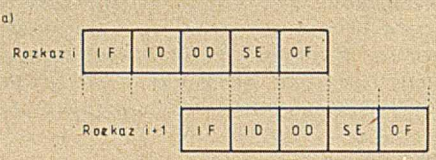
Mechanizm opóźnionego skoku, podobnie jak w innych maszynach z przetwarzaniem potokowym, wpływa na efektywność wykonywania kodu z dużą liczbą różnego rodzaju skoków, jaki powstaje z kompilacji programów napisanych w językach wysokiego poziomu (konstrukcje CALL, RETURN, IF, WHILE, CASE itd.).
Implementacja RISC II zawiera 41 tys. tranzystorów.

MIPS

Celem projektu MIPS [9, 13, 14, 15] podjętego w 1981 r. w Uniwersytecie Stanford była optymalizacja podziału funkcji między sprzęt i oprogramowanie stacji roboczej (ang. *workstation*) o wysokiej wydajności. Jak w innych komputerach RISC, rozkazy są proste i wykonują się w 1 cyklu. Potok rozkazów jest pięciostopniowy. Na podstawie wyników badań zrezygnowano z adresowania z dokładnością do bajtu, gdyż wymagało to istnienia bloku wyrównywania danych na czasowo krytycznej ścieżce między PAO a CPU. Oszacowania pokazały bowiem, że blok wyrównywania adresu powoduje straty czasowe sięgające 15–20%.

Wyróżniającą właściwością maszyny zawartą w akronimie jej nazwy /MIPS – *Microprocessor without Interlocked Pipeline Stages*) jest brak sprzętu chroniącego przed hazardami przy dostępie do obiektów przetwarzanych przez poszczególne stopnie potoku. Ciężar usuwania hazardów przeniesiono do tzw. *reorganizatora* – odrębnego programu odpowiedzialnego za modyfikowanie kodu wyprodukowanego przez kompilator (rys. 8). Reorganizator tworzy dwa umowne poziomy architektury:
- **asemblerowy**, na którym nie jest widoczna specyfika przetwarzania potokowego, a programista jest zwolniony od zajmowania się hazardami,
- **maszynowy**, na którym występują omówione wyżej problemy reorganizacji kodu.

Inną właściwością MIPS jest brak kodów warunków, które komplikują projektowanie układów VLSI, ponieważ wymagają nieregularnej struktury i złożonego dekodowania informacji o tym, które rozkazy i na jakie wskaźniki warunków mają wpływ. W zamian MIPS udostępnia rozkaz „porównaj i skocz", który wykonuje porównanie w czasie 1 cyklu.



b)
A : = B + C

c)
LOAD B, R1
LOAD C, R2
ADD R1, R2

STORE R2, A

d)
LOAD B, R1
LOAD C, R2
NOP
ADD R1, R2
STORE R2, A

Rys. 8. Potok w procesorze MIPS [1]
a) podział na stopnie, b) instrukcja źródłowa, c) przekład pierwotny, d) przekład po reorganizacji
Oznaczenia:
IF – sprowadzenie rozkazu,
ID – dekodowanie rozkazu, SE – zapamiętywanie lub wykonywanie,
OD – dekodowanie argumentu, OF – sprowadzanie argumentów

MIPS zaimplementowano w 1983 roku w technologii NMOS, co wymagało 24 tys. tranzystorów. Procesor miał cykl 250 ns.

Uniwersytecki projekt zaowocował powstaniem nowej firmy – MIPS Computer Systems, która wprowadziła na rynek procesor R2000.

R2000

Architektura R2000 [14] jest w pewnym stopniu wzorowana na MIPS. Potrojono liczbę rozkazów (z 31 do 102). Układ wykonano w technologii CMOS; z zegarem 16,7 MHZ osiąga on wydajność 10 MIPS. Mikroprocesor zawiera wewnętrzny układ MMU i zewnętrzną PAP o pojemności do 64 KB danych i (lub) 64 KB rozkazów. Istnieje oddzielny akcelerator zmiennoprzecinkowy R2010, realizujący standard IEEE 754 dla liczb 32- i 64-bitowych oraz format 80-bitowy.

Z użyciem R2000 MIPS produkuje pakiety oznaczane R2100, R2300, R2600 i R2065. Szczytowym osiągnięciem firmy jest komputer M/800 z pakietem R2600 o wydajności 6 MFLOPS: zawiera on 64 KB PAP dla danych i rozkazów, wirtualną przestrzeń adresową (4 GB) i pamięć fizyczną o pojemności 20 MB. M/800 pracuje pod kontrolą systemu UMIPS, zbliżonego do systemu Unix System V Rel. 3; udostępnia m.in. Pascal, C, Fortran 77, Adę, LPI-Cobol i LPI-PL/1. Procesor R2000 stosują również inne firmy do budowy stacji roboczych. W tabeli 5 przedstawiono porównanie wydajności komputerów firmy MIPS z komputerami VAX i Sun. Nowsze doniesienia [11,17] mówią o wprowadzeniu nowego modelu R3000 z koprocesorem R3010.

Tabela 5. Porównanie wydajności komputerów MIPS z VAX i Sun [14]

System	Wydajność względna
MIPS M/800 (R2600)	8,3
MIPS M/500 (R2300)	5,4
VAX 8650	4,8
VAX 8600	3,2
Sun 3/160 M	1,2
VAX-11/780	1,0

Spośród innych realizacji należałoby wspomnieć również o procesorach:
- SPARC (*Scalable Processor Architecture*) – firmy Sun [17],
- Force (*Forth Optimized RISC Computer Engine*) – Harris Semiconductor Corp. [15],
- MC 88000 – Motorola [17],
- Am 29000 – Advanced Micro Devices [10, 15, 17],
- NS 32523 – National Semiconductors,
- Transputer T800 – Inmos [2, 15, 16].

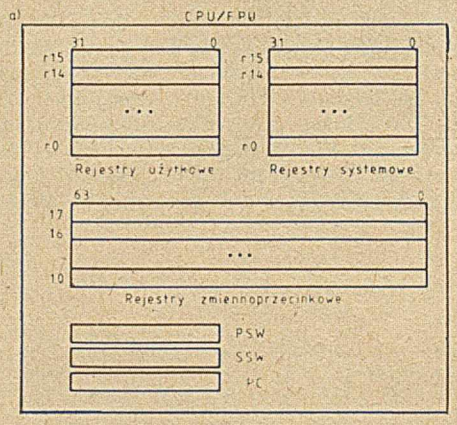
Cechy przypisywane komputerom o architekturze RISC ma m.in. mikroprocesor Clipper, deklarowany przez producenta wprost jako konstrukcja RISC.

Clipper

Clipper, 32-bitowy mikroprocesor o symbolu C100, został opracowany przez firmę Fairchild. Łączy on w sobie cechy maszyn RISC i CISC [1,6].

CPU udostępnia 32 rejestry 32-bitowe (po 16 dla programu użytkowego i systemu operacyjnego), 8 rejestrów zmiennoprzecinkowych, słowo stanu programu, słowo stanu systemu i licznik rozkazów (rys. 9). Przewidziano 10 formatów danych, w tym bajt, półsłowo, słowo, długie słowo – każde ze znakiem lub bez – oraz długą i krótką liczbę zmiennoprzecinkową. Rozkaz może mieć od 0 do 2 argumentów, z których co najwyżej jeden dotyczy pamięci. Udostępniono 9 trybów adresowania.

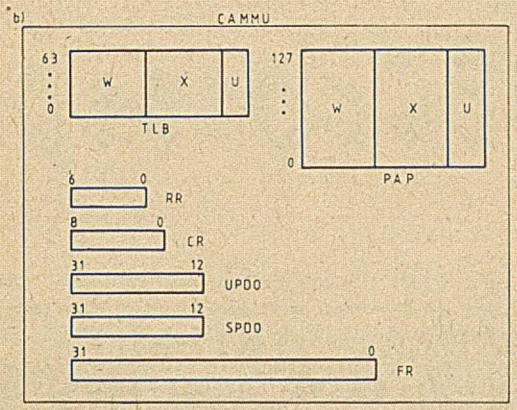
Moduł obliczeniowy Clippera składa się z 3 układów scalonych VLSI: CPU oraz dwóch identycznych układów CAMMU (ang. *Cache And Memory Management Unit*), odrębnych dla rozkazów i danych (rys. 9).



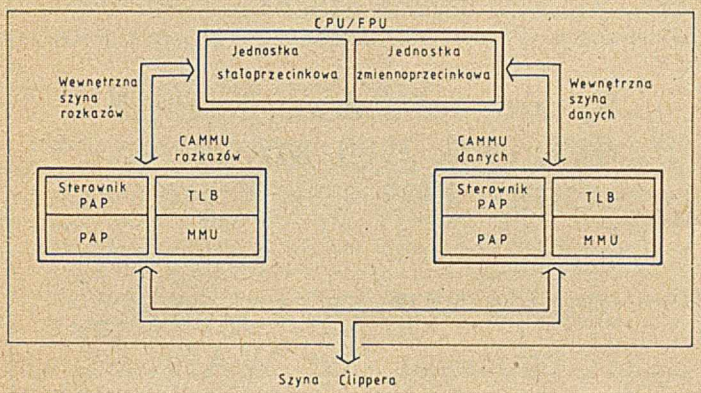
Rys. 9a. Rejestry programowe modułu Clipper – procesor z jednostką zmiennoprzecinkową [16]
Oznaczenia:
PSW – słowo stanu programu,
SSW – słowo stanu systemu,
PC – licznik rozkazów

CPU zawiera 3-stopniowy potok do przetwarzania danych całkowitych oraz wbudowaną jednostkę zmiennoprzecinkową realizującą operacje na liczbach 32- i 64-bitowych, zgodnie ze standardem IEEE 754; obydwie jednostki funkcjonalne działają współbieżnie. CAMMU za-

wiera w jednym układzie scalonym zarówno pamięć pomocniczą (*cache*), jak i bufor translacji adresów stron (TLB) wraz z buforami szyn i logiką sterującą.



Rys. 9b. Rejestry programowe modułu Clipper – CAMMU [16]
Oznaczenia:
TLB – bufor translacji,
PAP – pamięć pomocnicza,
UPDO – wskaźnik do katalogu tablic stron dla trybu użytkowego,
SPDO – wskaźnik do katalogu tablic stron dla trybu systemowego,
RR – rejestr zerowania,
CR – rejestr sterujący,
FR – rejestr błędu



Rys. 10 Struktura modułu obliczeniowego Clippera [16]

Clipper C100 jest dostępny w postaci modułu obliczeniowego o wymiarach 7,5 × 11,25 cm, zawierającego CPU, 2 układy CAMMU, układ zegarowy z kwarcem 66,6 MHz; moduł jest wyposażony w złącze krawędziowe 96-stykowe umożliwiające połączenie z resztą systemu. Złącze to udostępnia 32 linie adresu (danych), 8 linii przerwań wektorowych, 6 linii określających typ wykonywanej operacji (czytanie, zapis oraz długość danych), 3 linie określające wybrany obszar pamięci realnej (*sstem tag*) oraz wiele sygnałów sterujących przerwaniami i arbitrażem szyny. Cykl procesora wynosi 30 ns, średnia szybkość przetwarzania – 5 MIPS. Nowsza wersji C300 pracuje z zegarem 50 MHz [12]. Układy Clippera wytwarzane są przez firmę Intergraph. Moduł Clippera znalazł zastosowanie m.in. w pakiecie Panther-25 dla PC/AT z MS-DOS wytwarzanym przez firmę Spec Software AG, który zamienia ten komputer w supermini o wydajności 5 MIPS (2 MFLOPS), przy zegarze 25 MHz. Inna wersja, Panther-33, przy zegarze 33 MHz osiąga moc 8 MIPS (2,6 MFLOPS).

Wspólne cechy komputerów RISC

Zmniejszenie luki semantycznej jest osiągane przez zwiększenie wydajności komputera przy wykonywaniu programów napisanych w językach wysokiego poziomu, przy zasłonięciu przed programistą architektury poziomu języka maszynowego.

Zbiór rozkazów jest wynikiem dogłębnych badań kodu generowanego przez kompilatory; rzadziej używane rozkazy są składane z rozkazów podstawowych. Zbiór rozkazów komputerów jest ukierunkowany na wykonywanie skompilowanego kodu. Ponieważ poziom języka maszynowego jest dla programisty niedostępny, a kompilator i optymalizator kodu są traktowane jako integralne składniki systemu, komputer RISC może być zaliczony do maszyn wysokiego poziomu.

Poszczególne realizacje techniczne były ukierunkowane na indywidualne cele i poddane określonym ograniczeniom, można jednak wskazać wspólne cechy definiujące klasę RISC:

- ograniczenie kontaktu z PAO, przez konsekwentne stosowanie w modelu obliczeń rejestr–rejestr,
- dążenie do wykonania rozkazów w jednym cyklu maszynowym,
- potokowa realizacja rozkazów,
- układowa (najczęściej) ochrona przed hazardami,
- układowe (najczęściej) sterowanie,
- mała liczba rozkazów (do 100, rzadziej do 150),
- mała liczba trybów adresowania (1–2, rzadziej do 4),
- mała liczba formatów rozkazów (1–2, rzadziej do 4),
- wspomaganie realizacji programów w językach wysokiego poziomu.

W tabeli 6 zestawiono zalety i wady tej koncepcji.

Koncepcje pokrewne

W 1987 pojawiła się nowa idea – architektury o zbiorze rozkazów definiowanym przez użytkownika WISC (ang. *Writeable Instruction Set Computer*) [7, 8], łącząca zalety CISC i RISC. Komputer o takich

Tabela 6. Ocena elementów architektury RISC [9]

Właściwość	Zalety	Wady
Krótki cykl	Szybkie wykonywanie rozkazów	Ograniczony czas na dekodowanie i adresowanie rejestrów
Rozkazy jednocyklowe	Szybkie wykonywanie rozkazów	Wymagany potok i uproszczony zbiór rozkazów
Model obliczeń rejestr–rejestr	Uproszczenie zbioru rozkazów, potok	Ograniczenie złożonych operacji
Duży zbiór rejestrów	Pamięć wbudowana o szybkości ALU, wywoływanie podprogramów bez dostępu do pamięci operacyjnej	Zwiększenie czasu dostępu do rejestrów w cyklu ALU, duży narzut na przełączanie kontekstu
Pamięć pomocnicza	Szybki dostęp do rozkazów i danych, mechanizm dynamiczny	Ograniczona wielkość wbudowanej PAP, zewnętrzna PAP zwiększa czas dostępu
Odrębne PAP rozkazów i danych	Podwojenie przepustowości, dostęp w jednym cyklu	Wymagane dwie szyny lub przeplot na szynie
Opóźniony skok	Odzyskanie 60 – 80% cykli po rozkazie skoku, wyeliminowanie skomplikowanych zatrzymań potoku	Utrata do 30% miejsca po skoku, wymagany kompilator optymalizujący
Układowe sterowanie	Mniejsza logika, prosty, stały zbiór rozkazów, krótki cykl, jednocyklowe wykonanie	Nierealizowalność złożonych rozkazów, np. wielokrotnych operacji LOAD/STORE na rejestrach bez specjalnego automatu lub mikrokodu
Oparcie na oprogramowaniu kompilatora	Redukcja logiki sterującej, skrócenie cyklu, przesunięcie ciężaru optymalizacji wykonania na kompilator	Wymagany złożony kompilator optymalizujący, czasochłonne opracowanie kompilatora
Proste rozkazy o stałym formacie	Redukcja logiki sterującej, zmniejszenie cyklu	Nietolerowanie przypadków złożonych lub interpretacji w czasie wykonywania
MMU zintegrowane	Jednocyklowe przeglądanie TLB równocześnie z dostępem	Ograniczenie wielkości TLB, ograniczenie złożoności porównań
Orientacja na języki wysokiego poziomu	Optymalizacja dla wykonywania programów w językach wysokiego poziomu	Brak czytelnego języka poziomu asemblera, wymagany asembler optymalizujący

właściwościach dysponowałby możliwościami ładowania mikrokodu definiującego architekturę programową według uznania projektanta systemu.

Postulowanie właściwości architektury WISC są następujące:
- niekonieczne ograniczanie różnorodności rozkazów,
- niekonieczne ograniczanie się do modelu obliczeń na rejestrach, tj. do kontaktu z pamięcią jedynie przez rozkazy LOAD i STORE,
- stały format rozkazów,
- brak wymagań co do specjalnego kompilatora języka wysokiego poziomu,
- sprzętowa realizacja stosów do wywoływania podprogramów i przekazywania parametrów,
- pełne wykorzystanie przepustowości pamięci operacyjnej.

* * *

Lata osiemdziesiąte nie przyniosły zdecydowanego rozstrzygnięcia na korzyść żadnego z nurtów rozwoju architektury maszyn wysokiego poziomu. Oprócz wielu komercyjnych wyrobów o architekturze RISC oferujących wysoką moc obliczeniową, nadal silną pozycję zajmują uniwersalne maszyny CISC. Kierunek ten jest intensywnie rozwijany, zwłaszcza jeśli chodzi o architekturę ukierunkowaną na język. Wbrew zdecydowanym poglądom głoszonym przez przedstawicieli niektórych firm komputerowych (np. Sun), nie wydaje się, aby koncepcja RISC – mimo swoich ogromnych zalet – była zdolna wyprzeć CISC.

Wszystkie nowe realizacje techniczne mają postać niewielkiego zestawu (najczęściej jednego, rzadziej kilku) układów mikroprocesorowych. Poziom technologii układów scalonych będący w dyspozycji różnych firm jest w zasadzie podobny.

O technicznym sukcesie każdego nowego wyrobu, niezależnie od charakteru jego architektury, decydują przede wszystkim: trafne sformułowanie celu projektu, drobiazgowa analiza i wskazanie wąskich gardeł ograniczających wydajność systemu oraz umiejętne zastosowanie bardziej lub mniej znanych technik przyspieszania pracy.

LITERATURA

[1] Ackerman A., Baum G.: The Fairchild Clipper. BYTE, pp. 161–174, April 1987
[2] Caban D.: Transputer – narzędzie przetwarzania równoległego. Informatyka, Nr 6, s. 10–13, 1988
[3] Furht B.: A RISC Architecture with Two-Size Overlapping Register Windows. IEEE Micro, pp. 67–80, April 1988
[4] Hopkins M. E.: A perspective on the 801 / Reduced Instruction Set Computers. IBM. Journal, Vol. 26, No. 1, pp. 107–121, 1987
[5] How IBM designed its RISC-technology PC. Electronics (Электроника) No. 18. pp. 52–58, 1986
[6] Hunter C. B.: Introduction to the Clipper architecture. IEEE Micro. pp. 6–25, August 1987
[7] Koopman P.: The WISC Concept. A proposal for writeable instruction set computer. BYTE. pp. 187–194, April 1987
[8] Miller C.: CISC, RISC, WISC – What's in a name. IEEE Micro, February 1988
[9] Patterson D. A.: Reduced Instruction Set Computers. Comm. ACM, Vol. 28, No. 1, pp. 8–21, January 1985
[10] Robinson P.: How Much of a RISC. BYTE, pp. 143–150, April 1987
[11] Rowen C., Johnson M., Ries P.: The MIPS R3010 Floating-Point Coprocessor. IEEE Micro, pp. 53–62, June 1988
[12] Runyon S.: Second-generation Clipper hints a record 50 MHz. Electronics (Электроника) No. 23, 1987
[13] Silbey A., Milutinovic V., Mendoza-Grado V.: A survey of advanced microprocessors and HLL computer architectures. IEEE Computer, pp. 72–85, August 1986
[14] Tabak D.: RISC systems. Microprocessors and Microsystems. Vol. 12, No. 4, pp. 179–185, May 1988
[15] Weiss R.: RISC processors – the new wave in computer systems. Computer Design. pp. 53–73, May 15., 1987
[16] Wilson R.: British microprocessors – a lesson in innovation. Computer Design. pp. 32–42, November 1., 1988
[17] Wilson R.: RISC architectures take on heavyweight applications. Computer Design. pp. 59–79, May 15., 1988.

JAN MALEC
Instytut Cybernetyki Technicznej
Politechnika Wrocławska

Maszyny lispowe (1)

W ciągu ostatniego dziesięciolecia pojawiły się na rynku zachodnim wyspecjalizowane stacje robocze (ang. *workstation*), przeznaczone do przetwarzania symbolicznego w języku Lisp, nazywane maszynami lispowymi. Są one oferowane zarówno przez tak zwanych producentów sprzętu, jak Xerox czy Texas Instruments, jak i przez firmy wyspecjalizowane w produkcji tylko tego jednego rodzaju komputerów (Symbolica, LMI). Ponieważ ten rodzaj sprzętu nie jest szerzej znany w Polsce, warto przedstawić jego budowę (architekturę) oraz zastosowania.

Tekst podzielono na dwie części. W pierwszej opisano prototypy maszyn lispowych (ML). Dużo miejsca poświęcono pierwszemu, bez wątpienia najważniejszemu, a zarazem najlepiej udokumentowanemu procesorowi – CADR. Następnie opisano kilka prototypów o nieco inaczej rozwiązanej architekturze. Druga część dotyczy maszyn produkowanych seryjnie – omówiono tu maszyny firm Symbolics, LMI, Xerox i Texas Instruments.

PROTOTYPY MASZYN LISPOWYCH

Pierwsze prace nad prototypem maszyny lispowej rozpoczęto w MIT (*Massachusetts Institute of Technology*) na początku lat siedemdziesiątych pod kierunkiem R. Greenblatta. W roku 1976 powstał pierwszy działający procesor lispowy (CADR), a następnie niezależny komputer – stacja robocza – nazwana maszyną lispową MIT [10]. Osiągnięcie to dało impuls innym ośrodkom do podjęcia podobnych prac i już w roku 1979 Japończycy informowali o trzech niezależnych prototypach procesorów symbolicznych (por. [9, 15, 29, 23]).

W tym okresie zaprzestano publikowania informacji o dalszych pracach nad maszynami lispowymi, gdyż pojawił się chłonny rynek zbytu dla takich maszyn. Natomiast informacje o coraz intensywniejszych badaniach podstawowych nad architekturami ułatwiającymi obliczenia symboliczne w dalszym ciągu ukazują się w ogólnodostępnej literaturze. Należy tu zwłaszcza wymienić badania nad architekturą procesorów prologowych [14, 31, 32]. Dużym impulsem do badań w tym kierunku stał się japoński projekt komputera piątej generacji [11, 22]. Projekt ten kraje zachodu uznały za wyzwanie zarówno naukowe, jak i technologiczne; stał się on pierwszym z serii kilku międzynarodowych programów badań nad nową generacją komputerów.

W ostatnich latach w wielu ośrodkach są realizowane prace nad maszynami wieloprocesorowymi [6, 12, 27], pozwalającymi na zrównoleglenie dowolnego (w najogólniejszym przypadku) rodzaju obliczeń. Ich architektury odbiegają dość istotnie od koncepcji architektury komputerów piątej generacji. Prace te są w większości w fazie projektów, chociaż pewne z nich zaowocowały już produktami dostępnymi na rynku (chodzi tu zwłaszcza o transputery [20] opracowane przez brytyjską firmę Inmos oraz Connection Machine D. Hillisa [12] oferowaną przez założoną przez niego firmę).

Mówiąc o komputerach przeznaczonych do obliczeń symbolicznych, a w szczególności do obliczeń w Lispie, należy też wspomnieć o efektywnych interpreterach i kompilatorach pisanych na maszyny o klasycznej architekturze. W przypadku dużych komputerów, gdzie jest możliwa do osiągnięcia (a nawet przewyższana) efektywność ML, koszt pojedynczego stanowiska roboczego okazuje się zbyt duży (Lisp narzuca takie wymagania pamięciowe, że wielodostęp drastycznie obniża efektywność maszyny, a duży komputer dla pojedynczego, czy nawet kilku użytkowników, poza specyficznymi przypadkami, jest zbyt drogi). Rozwiązanie pośrednie, czyli typowa stacja robocza z odpowiednim oprogramowaniem (Lisp, Prolog), jest mniej więcej równoważne kosztem ML, efektywność jest natomiast niższa ze względu na niedostosowanie sprzętu do specyfiki tych języków. Najtańsze rozwiązanie – mikrokomputery klasy PC – z oczywistych względów nie może równać się efektywnością z maszynami lispowymi, chociaż ostatnio zrobiono pewien krok w kierunku przystosowania tego sprzętu do obliczeń symbolicznych. Polega on na dołączeniu do mikrokomputera dodatkowej karty zawierającej w sobie właściwie cały komputer lispowy (procesor, oprogramowanie w pamięci stałej, pamięć RAM o dużej pojemności) i przejmującej na siebie ciężar obliczeń, a pozostawiającej mikrokomputerowi rolę procesora wejścia–wyjścia i stacji pamięci masowej. Najbardziej znane rozwiązania to Hummingboard firmy Gold Hill Computers, z procesorem 80386, dołączana do mikrokomputerów klasy IBM PC/XT lub AT, oraz MicroExplorer firmy Texas Instrument, z kostką procesora lispowego tejże firmy, dołączany do nowszych wersji mikrokomputerów Macintosh.

Powyższe informacje wiążą się z kolejnym nurtem prac nad architekturami ML, a mianowicie nad próbami scalenia procesora lispowego. „Dojrzały" procesor lispowy w jednej kostce udało się uzyskać dopiero w drugiej połowie lat osiemdziesiątych projektantom firmy Texas Instruments (instalowany jest on w ML Explorer II), a obecnie słyszy się także o kostce lispowej firmy Symbolics.

CADR

Informacje podane poniżej są prawdziwe dla co najmniej trzech maszyn: MIT CADR, LMI CADR (pierwsza maszyna oferowana przez LMI) i LM-2 (pierwsza ML wyprodukowana przez Symbolics Inc.). Podstawą projektu ML zaproponowanej przez zespół z MIT były następujące wymagania:

- bardzo duża przestrzeń adresowa,
- architektura ułatwiająca operowanie adresami,
- szybka obsługa wywołania funkcji,
- współbieżność odzyskiwania pamięci (GC, ang. *garbage collection*) z obliczeniami podstawowymi.

Wymagania te wynikały z analizy pracy systemów języka Lisp na dużych komputerach, jedynych, na jakich w tym czasie można było implementować złożone systemy sztucznej inteligencji.

W trakcie trzyletniej pracy zespołu konstruującego ML udało się uzyskać komputer o następujących własnościach:

- duża (w r. 1978) przestrzeń adresowa (24 bity adresu),
- efektywność pamiętania danych i programów,
- duża liczba typów danych,
- jednorodne obszary typów danych (istotne przy GC),
- GC w czasie rzeczywistym,
- pojemna pamięć mikrokodu (16 K × 48 bitów),
- podprogramy w mikrokodzie,
- łatwy dostęp do poszczególnych bajtów informacji,
- szybkie wywoływanie funkcji,
- możliwość mikrokompilacji programów użytkownika,
- dobry edytor wbudowany w Lisp,
- dobry system operacyjny w Lispie,
- łatwość produkcji,
- prostota serwisowania,
- zgodność z magistralą Unibus firmy DEC.

Przyjęto następujące podstawowe rozwiązania sprzętowe: komórka listowa [1] składa się z dwóch słów po 32 bity, w każdym z nich 24 bity są przeznaczone na wskaźnik (adres), 5 bitów na znacznik (ang. *tag*) i 2 na kod rodzaju wskaźnika do pola CDR. Jednostka centralna jest mikroprogramowana, przy czym przyjęto 48-bitowy mikrorozkaz. Czas cyklu

[1] Komórka listowa jest podstawową strukturą wewnętrzną języka Lisp i zawiera dwa adresy – wskaźniki do innych struktur (komórek lub atomów); adresy te ze względów historycznych są nazywane CAR oraz CDR

procesora wynosi 180 ns. Na mikrokod przeznaczono 16 K słów pamięci RAM o szybkim dostępie. Szybki stos zajmuje 1 KB pamięci o czasie dostępu 200 ns, reszta stosu jest przechowywana w pamięci o czasie dostępu 1 μs. Pamięć RAM zajmuje 64 K słowa (MIT), czas dostępu do niej wynosi 1 μs. Późniejsze procesory (LMI CADR, LM-2) zawierały od 256 K do 4 M słów pamięci. Pamięć wirtualną zrealizowano na dysku o pojemności 16 M słów i czasie dostępu 25 ms. Urządzeniem wyświetlającym jest monitor graficzny, przy czym każdemu punktowi obrazu odpowiada jeden bit w pamięci. Obszar obrazu zajmuje 256 Kb.

Interpreter[2] został napisany dwupoziomowo: *APPLY*[3] dla funkcji skompilowanych (oraz funkcje podstawowe *CAR, CDR, CONS* itd.) napisano w mikrokodzie, natomiast już *READ, EVAL* i *PRINT* są napisane w Lispie. Lisp jest tu traktowany zarówno jak system operacyjny (podstawowa pętla *READ – EVAL – PRINT*), jak i język maszynowy komputera. Możliwe są odwołania zarówno Lisp → mikrokod, jak i mikrokod → Lisp. Dzięki wbudowaniu możliwości mikrokompilacji użytkownik może sam zdecydować, czy jego program ma w całości być pamiętany jako lista w Lispie, czy też w postaci modułów częściowo skompilowanych (w makrokodzie) a częściowo mikrokompilowanych (w mikrokodzie). Ze względu na 32-elementowy stos adresu mikroprogramu, system dopuszcza taką głębokość zagnieżdżeń wywołań w mikrokodzie.

Struktury danych

Przyjęte struktury danych umożliwiają implementowanie obszernej klasy algorytmów. Wszystkie dane są pamiętane wraz ze swoim typem (w polu znacznika), co pozwala na sprzętowe porównywanie typów, ewentualną ich konwersję, a także na wczesne i szybkie wykrywanie błędów. W system wbudowano następujące typy danych:

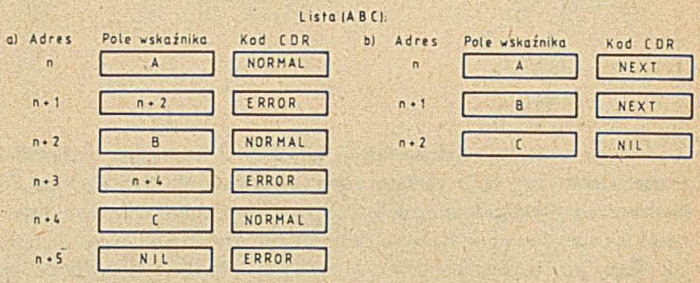
1. SYMBOLE (atomy) składające się z czterech słów:
PRINT NAME – wskazuje napis będący nazwą symbolu;
VALUE – zawiera bieżącą wartość symbolu (w ML MIT jest stosowane tzw. płytkie wiązanie zmiennych, ang. *shallow binding*);
FUNCTION – wskazuje typ funkcji określanej przez symbol oraz jej adres;
PROPERTY LIST – zawiera wskaźnik do listy właściwości.
Nazwą własną tego typu danych jest *DTP_SYMBOL*.

Rys. 1. Podział słowa pamięci na pola; x – pole znacznika używanego przez GC (1 bit), c – pole kodu cdr (2 bity), t – pole znacznika typu danej (5 bitów), p – pole wskaźnika (24 bity)

2. LISTY – nazwą tego typu danych jest *DTP_LIST*. Komórki listowe składają się z jednego lub dwóch słów 32-bitowych, o postaci jak na rys. 1. Pole kodu CDR pozwala w pewnych przypadkach dokonać kompresji obszaru zajmowanego przez listę, poprzez zapamiętanie kolejnych komórek listy w bezpośrednio po sobie następujących słowach pamięci (por. rys. 2).
Możliwe wartości pola kodu CDR są następujące:
CDR–NORMAL – komórka dwusłowowa;
CDR–NEXT – następne słowo jest następną komórką;
CDR–NIK – ostatnia komórka listy;
CDR–ERROR – ten adres nigdy nie może być prawidłowym adresem listowym[4].

3. LICZBY – w systemie wprowadzono dwa rodzaje liczb:
24-bitowe liczby całkowite ze znakiem (typ danych *DTP_FIX*), przy czym zawartość pola wskaźnika w słowie pamięci jest wartością liczby;
Pozostałe liczby (typ danych *DTP_EXTENDED_NUMBER*), czyli liczby zmiennoprzecinkowe, duże liczby całkowite, liczby zespolone.

4. MACIERZE – obszary bądź liczb całkowitych bez znaku o rozmiarach 1, 2, 4, 8, 16, 32 bitów, bądź znaków, każdy o rozmiarze 8 bitów. Typem danych macierzy jest *DTP_ARRAY*. W skrajnym przypadku obszary te mogą być jednowymiarowe (wektory), jednak system często korzysta z macierzy dwuwymiarowych. Pamięta się w nich m.in. obraz (ekran monitora wysokiej rozdzielczości), wewnętrzne tablice systemu (np. mapy pamięci), informacje o bieżących obliczeniach. Z każdą macierzą jest stowarzyszony tzw. nagłówek – wektor obiektów Lispowych, mający ustaloną przez użytkownika długość.



Rys. 2. Sposób pamiętania listy: a) klasyczny, b) skrócony A, B, C, NIL – adresy atomów o tych nazwach

5. REKORDY – istnieje możliwość definiowania struktur rekordowych, przy czym są one wewnętrznie zamieniane na macierze, nagłówki macierzy lub listy, w zależności od decyzji użytkownika.

PROGRAMY

Obliczanie programów może odbywać się jednym z trzech sposobów:

- przez interpretację struktury listowej zawierającej reprezentację programu:
- przez wykonanie skomplikowanego programu pamiętanego w makrokodzie. Programy te nie tracą informacji o konieczności sprawdzenia typów danych, obsługi nie podstawionych zmiennych lub sprawdzenia listy argumentów przekazanych do funkcji. Programy skompilowane są przechowywane w strukturach nazywanych ramkami wejść do funkcji (FEF, ang. *Function-Entry Frame*) o typie *DTP_FEF_POINTER*. FEF składa się z nagłówka, opisu argumentów, wskaźników do obiektów zewnętrznych i właściwego makrokodu;
- przez wykonanie mikroskompilowanego programu. System umożliwia bezpośrednią kompilację kodu Lispowego na mikrokod, co owocuje dużą szybkością działania programu i zmniejszeniem wielkości potrzebnej pamięci.

Podstawowym mechanizmem sterowania w CADR jest wysyłanie, (ang. *dispatching*), wykonywane na podstawie zawartości znacznika. Procesor ułatwia też wykonywanie instrukcji 16-bitowych (tzn. odpowiednich pól ze słowa), które powstają w wyniku kompilacji programu.

Wywołanie funkcji (instrukcja *CALL* makrokodu) powoduje otworzenie ramki wywołania na stosie (do 128 słów). Następnie procedura wołająca umieszcza argumenty na stosie w tych miejscach, w których wołana funkcja ich oczekuje. Pozwala to uniknąć zbędnego kopiowania argumentów. Wartość ostatniego argumentu jest umieszczana w polu *OSTATNI* ramki, co powoduje wykonanie wywołania.

Pamięć

Adresowalny obszar pamięci wynosi 16 M słów. Pamięć wirtualną zrealizowano na dysku stałym, tylko 64 K słowa informacji są przechowywane w pamięci RAM. Pamięć jest podzielona na 65 536 obszarów po 256 słów, przy czym część z nich (tzw. obszary statyczne), rzadko zmieniające swoją zawartość, nie są obsługiwane przez GC. Dla potrzeb GC wszystkie wskaźniki na zewnątrz obszarów statycznych są pamiętane w specjalnym „wektorze wyjść". Tłumaczenie adresu wirtualnego na adres fizyczny odbywa się przy użyciu dwupoziomowej mapy pamięci. Zastosowano kopiujący GC zaproponowany przez Bakera [3], pracujący w czasie rzeczywistym i nie blokujący komputera na dłuższe okresy.

FLATS

FLATS jest maszyną zaprojektowaną w Uniwersytecie Tokijskim [9], w latach 1978–1979, z myślą o obliczeniach w dwóch diametralnie różnych językach: Lispie i Fortranie. Dlatego ma ona nieco inną architekturę, niż typowe ML.

---

[2] Na temat sposobu obliczania programów zapisanych w Lispie, a w szczególności ich interpretacji, napisano wiele książek. Godne polecenia są [1, 2]; w języku polskim o mechanizmie obliczania w Lispie traktują książki [13, 19]

[3] Podstawowa procedura interpretera obliczająca wartość funkcji dla podanych argumentów

[4] Ponieważ wskaźnik do obszaru komórek listowych może pokazywać wyłącznie całą komórkę, a nie jej pole CDR (czyli wnętrze)

[5] Jest to zmienna, której wartością jest lista wszystkich znanych systemowi obiektów atomowych

Najważniejszymi cechami FLATS z punktu widzenia użytkownika są:
- nieograniczona precyzja obliczeń, zarówno w przypadku liczb całkowitych, jak i rzeczywistych;
- tabelaryzacja funkcji standardowych, znacznie przyspieszająca obliczenia numeryczne;
- pseudoasocjacyjna organizacja pamięci;
- wbudowane typy danych: łańcuchy, liczby całkowite, liczby zmiennoprzecinkowe, wektory, listy, zbiory.

Architektura komputera FLATS opiera się na podziale słowa maszynowego na część znaczącą dla obliczeń (dana lub wskaźnik) i część znaczącą dla sterowania (pole znacznika kodujące jeden z sześciu typów danych). Pamięć jest zorganizowana pseudoasocjacyjnie i obsługiwana (adresowana) pośrednio, przez sprzętowe tablice rozproszone. Pamięć jest adresowana 32 bitami (komórka listowa ma 64 bity) i jest podzielona na spójne obszary przeznaczone na ten sam typ bądź rodzaj danych.

W komputerze FLATS, w celu przyspieszenia obliczeń, zrealizowano sprzętowy stos (Lisp) i równolegle z nim 128 szybkich rejestrów ogólnego przeznaczenia (Fortran). Podstawowe operacje listowe (*car, cdr, cons, atom*) są wykonywane w jednym cyklu dostępu do pamięci. Także GC korzysta ze sprzętowo wspomaganych algorytmów zamiany wskaźników.

NK3

NK3 jest maszyną lispową zrealizowaną w 1979 roku w Uniwersytecie w Kyoto [23]. Jest to maszyna mikroprogramowana o słowie 32-bitowym, z którego 10 bitów jest przeznaczonych na znacznik. Pozostałymi 22 bitami można zaadresować do 4 MB pamięci. W trakcie prób używano pamięci o pojemności 512 KB i czasie dostępu 800 ns, dzielonej z minikomputerem Interdata 8/32, co znacznie spowalniało dostęp do pamięci (czas cyklu wynosił 2 μs) i obsługę urządzeń we-wy. Pamięć mikroprogramu zajmowała 4 K słowa 42-bitowe. W celu przyspieszenia obliczeń szczyt stosu zrealizowano sprzętowo (16 słów × 32 bity); był on obsługiwany asynchronicznie względem CPU. Skoki mikroprogramu były realizowane przez szybką tablicę przejść – pamięć 1 K słów 15 bitowych.

Oprócz standardowych funkcji Lispu 1.5, zaprogramowano też w mikrokodzie instrukcje *EXChange* (wymień) i *INITialize* (rozpocznij), pozwalające bezpośrednio implementować procesy współbieżne.

Ze względu na brak możliwości użycia podprogramów w mikrokodzie, rezultaty okazały się niewystarczające w stosunku do oczekiwań autorów. Pomimo tego osiągnięto czas obliczeń programów testowych tylko dwa razy dłuższy niż na największych komputerach japońskich (dokładniej na M190). Autorzy projektu przewidywali dalsze ulepszenia prototypu.

Eksperymentalna Maszyna Lispowa

Eksperymentalna Maszyna Lispowa (EML) z Uniwersytetu w Kobe [29] jest modułem dołączonym do szymy Unibus i współpracującym z procesorem LSI-11, który pełni funkcje inicjujące, sterujące i komunikacyjne w systemie.

Pamięć systemu ma 64 K słowa × 32 bity i czas dostępu mniejszy niż 75 ns. Pamięć mikrokodu zajmuje 4 K słowa × 56 bitów. Zaprojektowano specjalny sprzęt do wydobywania informacji z pojedynczego słowa pamięci (ang. *field extractor*). Przy adresowaniu zastosowano także specjalną (1 K × 3 bity) pamięć, służącą do tworzenia fizycznego adresu i przyspieszającą dostęp do typu danej (funkcja pola znacznik). Zastosowano szybki stos (o czasie dostępu 70 ns) o pojemności 4 K słów 16-bitowych.

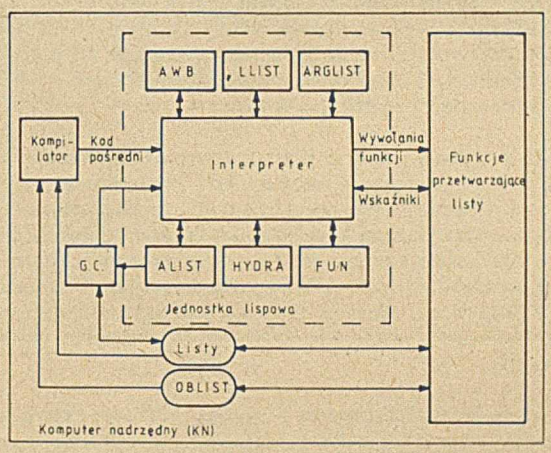
W maszynie zaimplementowano klasyczne typy danych: symbole atomowe, liczby stało- i zmiennoprzecinkowe oraz listy. Wyniki testów EML [29] oraz ich porównanie z wynikami uzyskanymi na komputerach kompatybilnych z IBM PC XT/AT zamieszczono w tabeli.

Porównanie czasu obliczeń przykładowych programów w EML (dane wg [29]) i w komputerach zgodnych z IBM PC/XT lub AT (1986)

Nazwa	Czas interpretacji programu			
	EML (ms)	XT (4,77 MHz) (s)	XT (8 MHz) (s)	AT (s)
TPU1	904	59	37	21
TPU2	3426	268	165	98
TPU3	1365	105	64	38
TPU4	1815	136	83	49
TPU5	238	25	15	9
TPU6	6728	567	345	204
TPU7		130	78	46
TPU8		118	71	42
TPU9		84	50	30

Mikroprogramowana Jednostka Lispowa

Mikroprogramowana Jednostka Lispowa powstała w latach 1980–1983 w uniwersytecie w Kaiserslautern [26]. Ideą tego rozwiązania jest podział obliczeń między dwie ściśle powiązane ze sobą maszyny: klasyczny komputer używany do operowania wskaźnikami (służy do tego 8-bitowy mikrokomputer) oraz dołączoną do niego mikroprogramowaną jednostkę służącą do przetwarzania struktur sterowania języka Lisp. Podziału programów na operacje manipulacji wskaźnikami i operacje przekazujące sterowanie dokonuje kompilator tworzący tzw. kod pośredni, przekazywany następnie do jednostki lispowej (JL, por. rys. 3).



Rys. 3. ML z Kaiserlautern – diagram przepływu danych

Istnieją dwa sposoby obliczania skompilowanego wyposażenia w Lispie (tzn. kodu pośredniego): maszyna rejestrowa o architekturze pokazanej np. w [1] oraz maszyna stosowa, zaproponowana w [16]. W opisywanym rozwiązaniu przyjęto drugą z tych propozycji. Danymi przetwarzanymi w JL są 16-bitowe słowa kodu pośredniego przesyłane między kilkoma stosami, z i do komputera nadrzędnego oraz do jednostki arytmetyczno-logicznej. Jak z tego wynika, mikroprogram JL steruje pamięciami stosowymi, ALU oraz portami komunikacyjnymi.

Do zadań komputera nadrzędnego KN należą wszystkie operacje przetwarzania list: *car, cdr, cons, get, read, print*, itd. Kompilator, tłumaczący postać listową programu na kod pośredni stanowiący wejście do JL, także rezyduje w KN. Trzecim modułem programowym KN jest GC. Dodatkowo, w KN są pamiętane zmienne systemowe (w tym *OBLIST*[5]) i listy.

JL ma sześć stosów (por. rys. 3), każdy do pamiętania innego rodzaju informacji. *AWB* zapamiętuje listę poleceń podanych w kodzie pośrednim, *FUN* listę funkcji do wykonania przez KN, *ARGLIST* argumenty tych funkcji – wskaźniki do list zapamiętanych w KN, *LLIST* pamięta *LAMBDA* – wyrażenia odpowiadające programom w *AWB*, *ALIST* jest środowiskiem obliczeń, a *HYDRA* zawiera bieżące wartości wszystkich elementów zmiennej *OBLIST*, w celu łatwej implementacji płytkiego wiązania zmiennych.

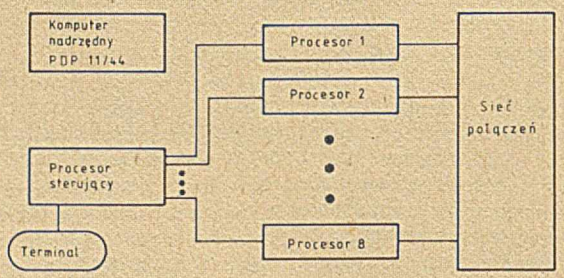
Po skompilowaniu wczytanego programu sterowanie jest przekazywane do JL, która z kolei odwołuje się do KN wywołując funkcje

operujące listami. Równocześnie może działać GC, co stanowi dodatkową korzyść.

Zaimplementowanym językiem jest Lisp 1.5.

EM-3

EM-3 jest prototypem ML skonstruowanym w Laboratorium Elektrotechnicznym ETL Ibaraki [33]. Jest ona przykładem kolejnej generacji prototypów, wykorzystujących propozycje i doświadczenia najnowszych badań nad nowymi architekturami komputerów. W tym przypadku przyjęto paradygmat sterowania danymi – procesory obliczające są uaktywniane w zależności od aktualnego stanu procesu obliczeniowego[6].



Rys. 4. Architektura maszyny EM-3

EM-3 jest maszyną wieloprocesorową o architekturze pokazanej na rys. 4. Każdy z procesorów jest zbudowany przy użyciu mikroprocesora MC 68000 oraz specjalizowanego sprzętu służącego do komunikacji z pozostałymi procesorami. Do komunikacji z procesorem sterującym służy 16-bitowy port równoległy.

Dane do przetworzenia oraz pośrednie wyniki przetwarzania krążą w sieci w postaci pakietów, tj. bloków informacji, którą należy dalej przetworzyć. Koncepcja sterowania danymi została tu wykorzystana następująco: wprowadzono kilka rodzajów pakietów, tworząc hierarchię odpowiadającą stopniowi przetworzenia informacji. Są to **wyniki częściowe, pseudowyniki, półwyniki i wyniki rzeczywiste.** W zależności od rodzaju wykonywanej operacji, procesor jest uaktywniany wówczas, gdy dysponuje argumentami o wymaganych rodzajach (stopniach) przetworzenia.

Używanym w EM-3 dialektem Lispu jest EMLisp – język w pełni funkcjonalny, umożliwiający wprowadzenie klarownej strategii obliczania. Od Lispu różni go odrzucenie wszelkich konstrukcji iteracyjnych (*PROG, GO, DO, LOOP* itp.), macierzy, (*ARRAY, AGET*, ...), modyfikatorów list (*RPLACA, RPLACD, NCONC*, ...) oraz dodanie funkcji blokującej *BLOCK* i równolegle wyliczanego wyrażenia warunkowego *PCOND*.

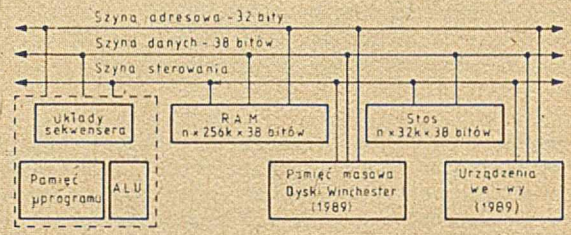
Obliczanie programu polega na rozesłaniu go przez procesor sterujący do poszczególnych procesorów. Jedną z podstawowych możliwości równoleglenia obliczeń jest etap obliczania wartości argumentów funkcji więcej niż jednej zmiennej. Niektóre z funkcji mogą być obliczane (przynajmniej do pewnego momentu) bez znajomości wszystkich swoich argumentów – tu właśnie wykorzystuje się możliwości oferowane przez hierarchię wyników częściowych.

Analiza symulacyjna wykazała, że powyższa architektura i przyjęty sposób obliczania pozwalają w niektórych przypadkach (zadania testowe: ustawianie hetmanów na szachownicy, szybkie sortowanie, obliczanie wyrazów ciągu Fibonacciego) na pięciokrotne skrócenie czasu obliczeń (przy 8 procesorach) w porównaniu z obliczeniami z użyciem jednego procesora.

Polska Maszyna Lispowa

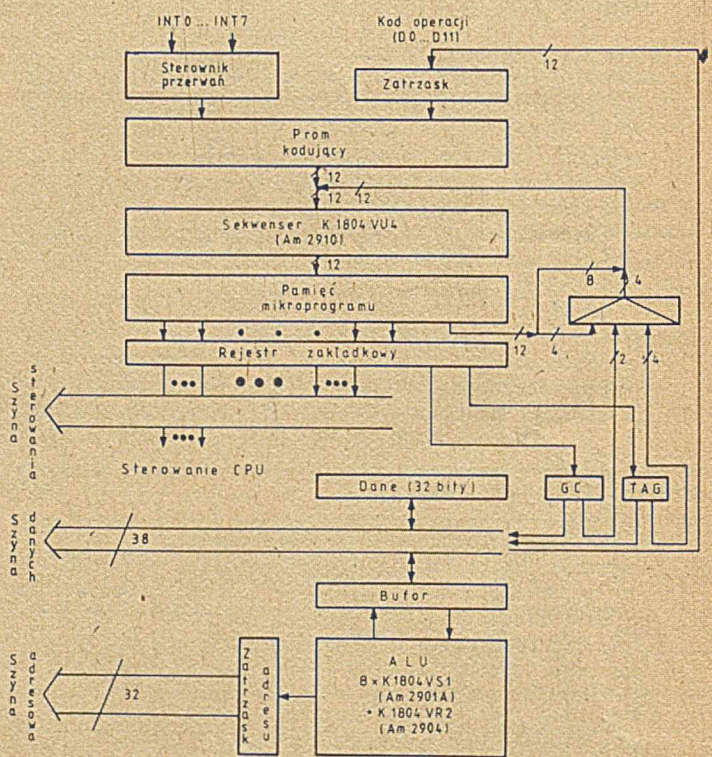
Od roku 1986 w Instytucie Cybernetyki Technicznej Politechniki Wrocławskiej trwają prace nad prototypem procesora ML [17]. Zaproponowane przez nas rozwiązanie (por. rys. 5) charakteryzuje się następującymi cechami:

- jednostka centralna komputera jest mikroprogramowana,
- mikroprogram może rezydować zarówno w pamięci ROM, jak i RAM (ale bez możliwości mikrokompilacji),
- czas cyklu zegara wynosi maksymalnie 220 ns,
- słowo danych jest podzielone na znacznik i wskaźnik,
- pamięć RAM danych zajmuje co najmniej 512 K słów obszaru adresowalnego (z możliwością dobudowania pamięci notatnikowej),
- istnieje tylko jeden stos w systemie,
- pamięć masowa (dysk Winchester) jest obsługiwana przez kanał DMA,
- układy LSI użyte w projekcie są produkowane w krajach RWPG,
- prototyp może być skonstruowany w ciągu najbliższego roku kalendarzowego (bez oprogramowania).



Rys. 5. Schemat blokowy architektury ICT ML

Zaprojektowano jednostkę centralną (rys. 6) zrealizowaną z użyciem mikroprocesorów segmentowych serii KR1804, produkcji ZSRR (odpowiednikiem jest seria 2900 firmy AMD). Zaproponowana architektura nie odbiega w zasadzie od koncepcji zastosowanych już w ML MIT. Projekt wstępny, zakładający cykl pracy procesora ok. 300 ns pozwala skorzystać z tanich pamięci RAM i ROM realizowanych w technologii MOS. W celu przeanalizowania trybów dostępu do stosu zaproponowano tylko jeden stos, dostępny dla wszystkich rodzajów danych. Przyjęto słowo 40-bitowe, przy czym znaczniki zajmują 8 bitów. Mikroinstrukcja ma długość 88 bitów za równoległym układem makrorozkazu. ALU przetwarza słowa 32-bitowe. Pamięci: mikroprogramu i stosowa zajmują każda 8 K słów, a pamięć RAM – 512 K słów.



Rys. 6. Schemat blokowy CPV

[6] Dokładniej, w zależności od tego, czy dane potrzebne w danym kroku obliczeń są już dostępne, czy jeszcze nie

LITERATURA

[1] Abelson H., Sussman G. J.: Structure and Interpretation of Computer Programs. The MIT Press, Cambridge (MA), 1985

[2] Allen J. R.: Anatomy of Lisp. McGraw-Hill (NY), 1978

[3] Baker H. G.: List Processing in Real Time on a Serial Computer. Communications of the ACM, 21(4), 1978

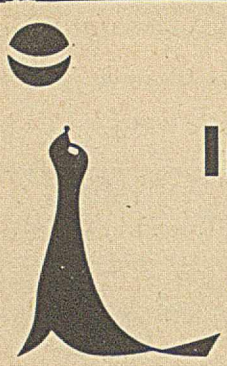
[4] Bobrow D., Wegbreit B.: A Model and Stack Implementation of Multiple Environments. Communications of the ACM, 16(10), pp. 591–603, 1973
[5] Bobrow D., Clark D.: Compact Encodings of List Structure. ACM Trans. on Prog. Lang. and Systems, 1(2), 1979
[6] Davis A. L., Robinson S. V.: The Architecture of the FAIM-1 Symbolic Multiprocessing System. Proc. 9th IJCAI, Los Angeles, pp. 32–38, 1985
[7] Deutsch L. P., Bobrow D.: An Efficient, Incremental, Automatic Garbage Collector. Communications of the ACM, 1976
[8] Gabriel R. P.: Performance and Evaluation of Lisp Systems. The MIT Press, Cambridge (MA), 1985
[9] Goto E., et al.: FLATS, a Machine for Numerical, Symbolic, and Associative Computing. Proc. 6th IJCAI, Tokyo, pp. 1058–1066, 1979
[10] Greenblatt R., et al.: The Lisp Machine. In: Artificial Inteligence – An MIT Perspective. P. H. Winston, R. H. Brown (Eds.), The MIT Press, Cambridge (MA), pp. 345–374, 1979
[11] Hertzberger L. O.: The Architecture of Fifth Generation Inference Computers. Fifth Generation Computer Systems, 1(1), pp. 9–21, 1981
[12] Hillis D.: The Connection Machine. The MIT Press, Cambridge (MA), 1986
[13] Jurkiewicz Z., Lao M. J.: Programowanie w Lispie. PWN, Warszawa (w druku)
[14] Kaneda Y., et al.: Sequential Prolog Machine PEK. New Generation Computing, No.4, pp. 51–66, 1986
[15] Kurokawa T.: LISP Activities in Japan. Proc. 6th IJCAI, Tokyo, pp. 502–504, 1979
[16] Landin P., w: Programming Languages, Information Structures, and Machine Organization, McGraw-Hill, (NY) 1968
[17] Malec J: Projekt konstrukcji komputera lispowego. Raport ICT PWr., PRE 104/87, Wrocław, 1987
[18] Malec J.: Komputery do przetwarzania symbolicznego w języku Lisp. Materiały 2 Krajowej Konferencji Robotyki, Prace Naukowe ICT PWr., Seria Konferencje, No.34, pp. 133–143, 1988
[19] Martinek J.: Lisp – opis, realizacja, zastosowanie. WNT, Warszawa, 1979
[20] May D., Shepherd R.: The Transputer Implementation of OCCAM. Proc. FGCS '84, ICOT, Tokyo, pp. 533–541, 1984
[21] Moon D.: Symbolics Architecture. IEEE Computer, pp. 43–52, Jan. 1987
[22] Moto-Oka T., et al.: Challenge for Knowledge Information Processing Systems. Proc. Int. Conf. Fifth Generation Computer Systems, Tokyo, pp. 3–89, 1981
[23] Nagao M., et al.: Lisp Machine NK3 and Measurement of its Performance. Proc. 6th IJCAI, Tokyo, pp. 625–627, 1979
[24] Paluszyński W.: Overview of The Dorado Personal Computer. Rękopis niepublikowany, Seattle, 1983
[25] Productivity Plus... LISP Machine Inc., Los Angeles, 1984
[26] von Puttkamer E.: A Microprogrammed Lisp Machine, Microprocessing and Microprogramming, No.12, 9–14, 1983
[27] Shrobe H.: Symbolic Computing – Where Have We Been? Ehere Are We Now? Where Are We Going? Niepublikowane materiały referatu wygłoszonego podczas AAAI-86, Philadelphia (PA), 1986
[28] Steele G. L., Sussman G. J.: Desing of a Lisp-based Microprocessor. Comm, ACM, No.11, pp. 628–645, 1980
[29] Taki K., Kaneda Y., Maekawa S.: The Experimental Lisp Machine. Proc. 6th IJCAI, Tokyo, pp. 865–867, 1979
[30] Texas Instruments EXPLORER™ Technical Summary. Texas Instruments, Austin (TX), 1985
[31] Tick E.: An Overlapped Prolog Processor. SRI International, AI Center Technical Note 308, Menlo Park, Oct. 1983
[32] Warren D. H. D.: An Abstract Prolog Instruction Set. SRI International, AI Center Technical Note 309, Menlo Park, October 1983
[33] Yamaguchi Y., Toda K., Herath J., Yuba T.: EM-3 – A Lisp-Based Data-Driven Machine, Proc. FGCS '84, ICOT, Tokyo, pp. 524–532, 1984.

Docierają do redakcji sygnały z całego kraju o trudnościach z dostaniem INFORMATYKI w kioskach Ruchu.

Pragniemy wyjaśnić Czytelnikom przyczynę tych kłopotów. Redakcja podjęła decyzję zmniejszenia liczby egzemplarzy skierowanych do sprzedaży kioskowej, gdyż Ruch pobiera bardzo wysoką prowizję za swą usługę, nie gwarantując prawidłowego rozdziału egzemplarzy w sieci kioskowej.

W związku z tym prosimy wszystkich zainteresowanych nabyciem INFORMATYKI o kontakt z redakcją (tel. 39-14-34) lub Działem Handlowym SIGMY (tel. 40-30-86) – będziemy Państwu wysyłać egzemplarze za zaliczeniem pocztowym. Czytelnicy z terenu Warszawy mogą kupić poszczególne numery w Klubie SIGMY przy ul. Mazowieckiej 12, w księgarni PP „Domu Książki" przy ul. Mokotowskiej 51/53 oraz w lokalu redakcji.

Jednocześnie przypominamy, że najpewniejszą formą otrzymywania INFORMATYKI jest prenumerata. Szczegóły jej zamawiania na II stronie okładki.



INTERSOFT Sp. z o.o.

Nasze Biuro Handlu Zagranicznego poleca swoje usługi w zakresie exportu-importu każdego towaru. Prowadzi sprzedaż profesjonalnego sprzętu komputerowego i peryferii.

1. JĘZYKI PROGRAMOWANIA, BIBLIOTEKI, GRAFIKA	
Basic – programowanie w języku Basic cz. III	52 000 zł
Turbo Basic	183 000 zł
Proffesional Fortran + dodatki	144 000 zł
Fortran 77	132 600 zł
Fortran 80	65 300 zł
Język C – zastosowanie dla zaawansowanych	214 200 zł
Kompilator języka C	106 000 zł
System Turbo C V.1.5	79 000 zł
Turbo C biblioteka	150 000 zł
Turbo C podręcznik użytkownika	109 000 zł
Turbo Pascal	123 500 zł
Turbo Pascal V.4.0	277 000 zł
Turbo Pascal V.5.0	161 000 zł
Turbo Pascal V.5.5 dodatek do T.P.V.5.0	56 000 zł
Turbo Power Tools Plus	124 000 zł
Turbo Database Toolbox	33 000 zł
Modula 2 86	125 000 zł
Turbo Pascal 5.0 opis języka	189 000 zł
Turbo Pascal 4.0 (aut. Jan Bielecki)	153 000 zł
2. BAZY DANYCH, PAKIETY ZINTEGROWANE	
Clipper do dBase III	30 000 zł
Clipper do dBase III +	131 000 zł
Clipper 86	158 000 zł
Clipper 87	212 700 zł
Clipper 87 do dBase III +	214 000 zł
dBase II i III – podręcznik dla zaawansowanych	205 000 zł
dBase III + instalacja pakietu relacyjnej bazy danych	22 000 zł
dBase III + zastosowanie	205 000 zł
dBase III + opis pakietu sieciowego	55 000 zł
dBase III + programowanie	126 000 zł
dBase III + poznawanie	111 000 zł
dBase III – poradnik encyklopedyczny	237 000 zł
dBase IV	249 000 zł
Framework II P	162 000 zł
Informix	261 000 zł
Lotus 1–2–3 + dodatek (komplet)	113 000 zł
Multiplan	66 000 zł
Pakiet C-ISAM	34 700 zł
CX-Fortran kompilator języka do systemu DOS 2	114 300 zł
Plib 86/Plink 86 opis języka	47 0000 zł
Statgraphics	91 000 zł
Programowanie systemów baz danych języka Clipper	107 000 zł
Kompilator CBasic tom. II	92 000 zł
3. SYSTEMY OPERACYJNE, PROGRAMY UŻYTKOWE	
System operacyjny DOS cz. 1 i 2	155 000 zł
System operacyjny DOS 3.10	47 00 zł
System operacyjny DOS 3.20	181 000 zł
System operacyjny DOS 3.30	180 000 zł
System operacyjny DOS 4.0	202 000 zł
Eureka	64 000 zł
Norton Commander	63 000 zł
Polywindows	50 000 zł
Sidekick	44 000 zł
4. EDYTORY	
Chl-2 Writer	31 000 zł
Personalny Edytor PE	68 300 zł
WordStar 2000	142 000 zł
5. CAD – SIECI	
AutoCad 2.17	137 000 zł
AutoCad	30 600 zł
Lanlink 4.0 sieć lokalna	80 000 zł
NOWELL, instalacja sieci Arcnet	9 000 zł
NOWELL, instalacja sieci	65 000 zł
NOWELL, podręcznik użytkowania sieci	116 000 zł
Or-Cad	134 000 zł
6. ROZMAITOŚCI	
Przewodnik programisty	205 000 zł
Printer STAR NL-10 instrukcja	20 000 zł
Wprowadzenie do użytkowania komputerów osobistych klasy IBM PC	43 000 zł
Drukarka NX-16 instrukcja	106 000 zł
Drukarz	14 000 zł
Karta – system rozliczania kart drogowych	10 000 zł
Appsgen	18 000 zł

Zamówienia prosimy kierować pod adresem:
INTERSOFT Sp. z o.o.
00–496 Warszawa, ul. Krucza 16/22
telefony: 28-44-81 w. 284, 352; 28-29-53
teleks: 812414

EO/232/89

PIOTR DEMBIŃSKI
Instytut Podstaw Informatyki PAN
Warszawa

Protokoły komunikacyjne (1)

Rozwój techniki obliczeniowej można dzielić na różne okresy oraz wskazywać różne wydarzenia i okoliczności, które decydowały (lub współdecydowały) o tendencjach lub przyjmowanych rozwiązaniach. Z punktu widzenia problematyki, która nas tutaj interesuje, warto chyba przypomnieć i uwypuklić zmiany zachodzące w stosunkach użytkownik-komputer.

Przez większość czasu dzielącego nas od pierwszych lat, w których komputery rozpoczęły odgrywać jakąś rolę w naszym życiu, pozostawały one dużymi, skomplikowanymi i drogimi maszynami zainstalowanymi w izolowanych pomieszczeniach, do których dostęp był zarezerwowany dla tajemniczych (i na ogół brodatych) ekspertów. W latach pięćdziesiątych użytkownicy (w tym również eksperci) pracowali w systemie wsadowym, w którym żadne oddziaływanie na programy w trakcie ich wykonywania nie było możliwe. Bezpośrednia komunikacja użytkownik-komputer nie istniała.

W latach sześćdziesiątych nastąpił pierwszy przełom w tej dziedzinie, spowodowany wprowadzeniem systemów wielodostępnych z podziałem czasu (ang. *time-sharing*), w których potężne komputery mogły jednocześnie obsługiwać wielu użytkowników komunikujących się z nimi – w sposób interakcyjny – za pomocą przyłączonych urządzeń końcowych (ang. *interactive terminals*). Użytkownicy otrzymali więc własny dostęp do komputera, na którym mogli pracować jednocześnie, każdy wykonując swoje własne zadanie. Centralnie zgromadzone w komputerze dane mogły być wspólnie wykorzystywane. Pojawiły się pierwsze rozwiązania komunikacyjne związane z połączeniem terminal-komputer. Gwałtownie zwiększyła się liczba zastosowań i użytkowników. Ciągle jednak przeważali eksperci, a co najmniej użytkownicy o pewnym profilu zawodowym.

Kolejny przełom nastąpił w latach siedemdziesiątych, wraz z wprowadzeniem minikomputerów i komputerów osobistych. Użytkownikiem stał się każdy; biura, urzędy i przedsiębiorstwa zapełniły się małymi urządzeniami liczącymi o mocy obliczeniowej często porównywalnej z dużymi jednostkami centralnymi z niedalekiej przeszłości. Komputer przeniesiony w miejsce pracy każdego użytkownika stał się jednym z jego podstawowych narzędzi działania, o łatwym, a nawet zachęcającym dostępie.

Efektywne wykorzystanie tej masy urządzeń, oprogramowania i zgromadzonych w różnych miejscach informacji, wymagało połączenia ich siecią komunikacyjną. W ten sposób ogromna liczba użytkowników mogła wykorzystywać potencjał obliczeniowy rozproszony w różnych miejscach tego samego budynku, w różnych filiach lub wydziałach tego samego przedsiębiorstwa, w wielu miastach, czy wreszcie krajach. W niepamięć odeszły rojenia o komputerze-gigancie, który pomieści „całą" dostępną informację i będzie obsługiwać miliony użytkowników. Nadeszła (a może raczej nadchodzi) era rozproszonego przetwarzania i przechowywania informacji, era, w której miliony użytkowników mają dostęp do tej informacji i rozproszonych mocy obliczeniowych za pomocą szybkich, niezawodnych i łatwo dostępnych środków komunikacji. Zapewnienie takich środków komunikacyjnych pozwala na elastyczne dostosowane do potrzeb, rozbudowywanie i zmienianie konfiguracji sieci.

W latach siedemdziesiątych powstało bardzo wiele tzw. sieci lokalnych o ograniczonym – np. do jednego budynku lub kompleksu budynków – zasięgu połączeń. Najbardziej znaną obecnie siecią lokalną jest Ethernet [3] firmy Digital Equipment Corporation (DEC). W tym też czasie zbierano doświadczenia z użytkowania pierwszej publicznej sieci Arpanet [2], łączącej początkowo wybrane uniwersytety amerykańskie. Powstawały inne sieci publicznego użytku, jak SNA (*System Network Architecture*) firmy IBM lub DECNET firmy DEC. Powstawały też węzły komunikacyjne zapewniające możliwość włączania sieci lokalnych do sieci o większym zasięgu, wykorzystujących łącza telefoniczne, satelitarne lub radiowe. Badano i mierzono efekty pracy węzłów komunikacyjnych w różnych konfiguracjach i przy różnych metodach dostępu do nich. Nastąpił rozwój burzliwy, ale też najczęściej nieskoordynowany. Producenci zazdrośnie strzegli swoich rozwiązań w nadziei, że zdołają uzyskać wyłączność swoich produktów. Problem to nie nowy, ale nie były to już lata sześćdziesiąte, w których niepodzielnie panował „rynek producentów", tj. taki, w którym użytkownik przyjmował z pokorą skutki walk konkurencyjnych kilku gigantów komputerowych. Wytworzyła się wyraźna presja na wprowadzenie rozwiązań ujednoliconych, znormalizowanych. Mówiąc krótko, użytkownicy mieli dość sytuacji, w której zainstalowany sprzęt wymagał kosztownego oprogramowania dostosowawczego za każdym razem, gdy chciało się go włączyć do współpracy z już istniejącym, np. innej firmy. Co gorsze, coraz częściej komputery wyprodukowane przez tę samą firmę nie umiały współpracować ze sobą bez takich specjalnych zabiegów adaptacyjnych. W tej sytuacji dojrzewała koncepcja ustalenia norm światowych. W rezultacie, na przełomie lat siedemdziesiątych i osiemdziesiątych powstał tzw. „Podstawowy Model odniesienia ISO OSI" (*International Standards Organization – Open Systems Interconnection – Basic Reference Model*) [1], który powoli i nie bez oporów, staje się punktem odniesienia w projektowaniu sieci i protokołów komunikacyjnych.

W tym miejscu należy zwrócić uwagę na to, że wprowadzenie norm w tak szybko i dynamicznie rozwijającej się dziedzinie wymaga niemałego wysiłku i pokonania wielu trudności. Jeszcze trudniej jest przekonać zainteresowanych, żeby zechcieli istniejących norm przestrzegać. Instytucje normalizacyjne mogą jedynie proponować swoje rozwiązania, ale nikt nie jest zobowiązany do ich stosowania. Wspomniane tutaj i zrozumiałe interesy producentów, koordynacja między instytucjami państwowymi różnych krajów, a także dostosowanie się do już istniejących standardów, np. w telekomunikacji, to tylko niektóre przeszkody, które należy pokonać. Trzeba też wspomnieć o pewnej konkurencyjności międzynarodowych organizacji normalizacyjnych. Z jednej bowiem strony ISO z drugiej CCITT (*Comité Consultatif Internationale Télégraphique et Téléphonique*) pretendują, każdy nie bez racji, do tworzenia norm w tej dziedzinie. Mimo dobrej woli obu organizacji powoduje to nieuniknione konflikty, dublowanie wysiłków – w sumie mnożenie trudności. Mimo to, a może właśnie dlatego wydaje się, że osiągnięto w tym przypadku sukces. Znormalizowana architektura sieci staje się (prawie) faktem.

WARSTWOWA ARCHITEKTURA SIECI

W projektowaniu sieci od początku stosowano zasadę ustalania hierarchii zadań, które prowadziły do podziału architektury każdego węzła sieci na tzw. **warstwy** (ang. *Mayers*). Warstwa niżej usytuowana w hierarchii oferuje usługi warstwie znajdującej się bezpośrednio powyżej niej. Ta ostatnia korzysta z oferowanych usług do zrealizowania swojego zadania.

Warto spróbować scharakteryzować taką warstwową organizację przez analogię z „przykładem z życia wziętym" (chociaż może z życia nieco wydumanego):

Dowódcy jednostek (warstwa 5) wymieniają między sobą b. ważne i b. tajne depesze. Każdy z nich korzysta z usług Biura Szyfrów (*BS* – warstwa 4), w którym tekst dostarczanej depeszy zostaje zaszyfrowany. Dobór szyfru zależy nie tylko od *BS* danej jednostki, ale także od zgody adresata (adresatów) depeszy i, być może, innych b. tajnych rozporządzeń. Cały ten kłopot z ustaleniem aktualnie obowiązującego szyfru *BS* rozwiązuje przez nawiązanie kontaktu z odpowiednimi *BS* innych jednostek, nie zaprzątając głowy dowódcom, którzy (jak to dowódcy) przygotowują inne b. ważne i b. tajne depesze. Dla tych

swoich własnych celów i dla ostatecznego przekazania treści depeszy, BS dołącza swe własne informacje i przekazuje tak uzupełnioną wiadomość do Biura Transportu (BT – Warstwa 3). Biuro Transportu ustala aktualne adresy odbiorców (dowódca przecież nie pamięta wszystkich adresów – zna tylko nazwy jednostek – a szyfranci znać ich nie powinni) i dorzuca je do otrzymanego z *BS* tekstu. *BT* ma też do swojej dyspozycji ścisłe instrukcje, które ustalają jakie węzły komunikacyjne należy omijać, jako podejrzane o wrogi sabotaż. Poza tym, zdecydowano (ku rozpaczy pracowników Biura), że *BT* ma wprowadzić własny system ochrony przed błędami takimi, jak: nieprawidłowa kolejność przesyłania depesz, gubienie ich po drodze lub dostarczanie dwa i więcej razy tej samej wiadomości. Mimo bowiem ciągłych zapewnień Biura Łączności (*BŁ* – warstwa 2) o 100% niezawodności jego usług, z których korzysta *BT*, zdarzają się tego rodzaju błędy bardzo denerwujące dowódców. Tak więc *BT* opatruje tekst otrzymany z *BS* całą masą dodatkowych informacji kontrolnych i dopiero taki pakiet przekazuje do *BŁ*, w którego gestii jest ustalenie dokładnej trasy przekazu i zapewnienie ochrony przed błędami transmisji. Znowu więc informacja przekazana przez *BT* zostaje zaopatrzona w dodatkowe elementy kontrolne i dopiero wtedy otrzymuje ją Departament Telekomunikacji (*DT* – warstwa 1), który zgodnie z instrukcjami dokonuje transmisji wybranym przez siebie kanałem komunikacyjnym. Przy odbiorze *DT* przekazuje otrzymane dane do *BŁ*, które sprawdza czy jest to ta wiadomość (np. co do kolejności), na którą czeka i czy można ją uznać za wolną od błędów. Jeżeli tak, to *BŁ* usuwa nagłówki kontrolne dotyczące komunikacji między BŁ-ami i następnie pozostałą część komunikatu przekazuje do *BT* wysyłając jednocześnie potwierdzenie odbioru do *BŁ* jednostki, która depeszę nadała. *BT* z kolei analizuje i usuwa nagłówki kontrolne swoich kolegów z nadającego *BT*. Jeżeli stwierdzi błąd (np. że nie do niego jest adresowana wiadomość), to wysyła przez swoje *BŁ* odpowiedź odmowy przyjęcia depeszy (np. z dopiskiem „adresat nieznany"). Jeżeli błędów nie ma, to uznaje, że tekst należy przekazać do *BS* do odszyfrowania. Wreszcie odszyfrowana depesza trafia na biurko dowódcy. W Centralnym Biurze Rozliczeń rozumiano dobrze konieczność wymiany b. ważnych i b. tajnych depesz, ale wielokrotnie zwracano uwagę na olbrzymie koszty komunikacji. Problem był trudny i delikatny ze względu na drażliwość dowódców, którzy mogliby źle przyjąć administracyjnie zarządzone ograniczenia. Problem doczekał się jednak zadowalającego rozwiązania dzięki pomysłowi skromnego szeregowca St. B., z jednego z Biur Łączności. Zauważył on mianowicie, że dowódcy mają tendencje do wysyłania depesz bardzo często (od tego są przecież dowódcami), ale bardzo krótkich (to cecha dobrych dowódców). Ponieważ najwięcej kosztuje samo połączenie, szeregowiec ST. B. wpadł na pomysł, by nie wysyłać każdej depeszy osobno, ale czekać jakiś czas aż uzbiera się ich więcej, łączyć je w zestawy o tym samym adresie i dopiero całe takie zestawy przesyłać. Otrzymane zestawy depesz będą rozdzielane ponownie w *BŁ* adresata. Ten prosty sposób poprawił znacznie wyniki ekonomiczne Centrali, a dowódcy, a także ich *BS* i *BT*, nie zauważyli nawet dokonanej modyfikacji.

Ten nieco swobodny przykład miał na celu wskazanie głównych cech i elementów charakteryzujących warstwowy model sieci. Omówiono je poniżej.

WARSTWY I ICH ZADANIA

Architektura każdej jednostki w sieci składa się z takiej samej liczby warstw i każda warstwa ma określone zadania do wykonania. Zadaniem np. Biura Szyfrów (*BS*) jest szyfrowanie depesz dowódcy własnej jednostki i odszyfrowywanie depesz nadchodzących z innych jednostek. Jednak nie ogranicza się ono do tego. *BS* ma także zadanie uzgadniania z *BS* każdego adresata, być może przy każdej depeszy, odpowiedniego szyfru oraz – na przykład – kontrolę jakości szyfrów, która może spowodować odrzucenie przez BS otrzymanej depeszy zawierającej błędy szyfrowania.

PROTOKÓŁ WARSTWY

Zadanie każdej warstwy wymaga jej komunikacji i dialogu z odpowiadającą jej warstwą innej jednostki. Ten dialog dotyczy spraw interesujących tę właśnie warstwę i jest niezależny od komunikatu, który dialog ten zainicjował. Taki komunikat, inicjujący dialog między warstwami tego samego poziomu, pochodzi (oprócz warstwy najwyższej) od warstwy bezpośrednio powyżej w hierarchii warstw, gdy jednostka pełni rolę nadajnika, lub od warstwy bezpośrednio poniżej (oprócz warstwy najniższej), gdy jednostka pełni rolę odbiornika. Ten właśnie bezpośredni dialog między tymi samymi warstwami jednostek składowych sieci stanowi **protokół** danej warstwy. Odbywa się on przez wymianę wiadomości oznaczonych *X.PDU* (ang. *Protocol Data Unit*), przy czym *X* jest nazwą warstwy.

PDU każdej warstwy *X* zawiera dane (ewentualnie przekodowane) warstwy bezpośrednio z nią sąsiadującej oraz **nagłówek** (ang. *header*), zawierający informacje właściwe dla protokołu warstwy *X*. Tak więc opis protokołu między Biurami Transportu w naszym przykładzie pomijałby całkowicie rolę Biur Łączności i Departamentów Telekomunikacji, a opis (części dotyczącej nadawania) protokołu wymiany depesz między dowódcami jednostek (warstwa najwyższa) mógłby brzmieć następująco:

1) wywołaj nazwę jednostki, do której nadana będzie depesza;
2) poczekaj *x* sekund na zgłoszenie się wywołanej jednostki i jeżeli nie otrzymasz zgłoszenia, to przejdź do 1 lub 5;
3) nadaj treść depeszy;
4) zakończ nadawanie i przejdź do 6;
5) anuluj nadawanie;
6) poczekaj *y* sekund na potwierdzenie przyjęcia depeszy lub anulowania nadawania i jeżeli nie nadejdzie ono w tym czasie, to przejdź do 3 lub 5 lub 7;
7) anuluj lub zakończ nadawanie nie czekając na potwierdzenie;
itd.

Opis podany w powyższych terminach stwarza wrażenie bezpośredniej komunikacji między dowódcami jednostek. Dowódca wie, że ma przekazać b. ważną i b. tajną wiadomość, lecz nie musiał nawet słyszeć o tym, że szyfrowanie służy utajnianiu przekazu, nie mówiąc już o rodzajach i zasadach działania technicznych środków komunikacji, będących w posiadaniu Departamentu Telekomunikacji lub o sposobach używanych przez Biura Łączności i Transportu zapewniających bezbłędny przekaz, uporanie się z problemem krzyżujących się wiadomości (ang. *colisions*) albo z tym, że jedne jednostki działają o wiele sprawniej i szybciej niż inne i te ostatnie są zawalone nadmiarem danych, których nie mogą przetworzyć (ang. *backpressure problem*). Co by to zresztą było, gdyby dowódca dowiedział się, że drogie urządzenia telekomunikacyjne psują się często i wtedy jego Departament Telekomunikacji po prostu posługuje się zwykłym posłańcem?!

Tak więc wymiana wiadomości typu *PDU* jest pewną „horyzontalną abstrakcją" rzeczywistości, mającą podkreślić fakt, że opis protokołu komunikacji dla danej warstwy powinien być niezależny od jakiejkolwiek jego implementacji.

USŁUGI WARSTWY I SPRZĘGI MIĘDZYWARSTWOWE

W rzeczywistości, i tak było w naszym przykładzie, komunikacja między tymi samymi warstwami różnych składowych sieci odbywa się (z wyjątkiem warstwy najniższej) przy użyciu **usług** (ang. *services*), oferowanych przez warstwy znajdujące się bezpośrednio pod nimi. Usługa formułuje zasady wymiany informacji i treść tej informacji między sąsiadującymi warstwami w jednostce składowej sieci. Inaczej mówiąc, usługi warstwy *X* charakteryzują głobalnie działanie warstw 1,..., *X* z punktu widzenia warstwy $X+1$, która z tych usług korzysta. Korzystanie z usług warstwy *X* przez warstwę $X+1$ polega na przekazywaniu i odbieraniu przez tę ostatnią wiadomości, oznaczanych *X.SDU* (ang. *Service Data Unit*), definiowanych przez warstwę *X* jako te, które odpowiednio ona sama może otrzymywać i wysyłać.

Jaki jest związek między *X.PDU* a *X.SDU*? Oczywiście *X.PDU* musi zostać, w jakiejś postaci przekazane niżej, ale *X.SDU* na ogół zawiera dodatkowe informacje i parametry. Te parametry i informacje mogą – na przykład – dotyczyć jakości usługi *QOS* (ang. *Quality of Service*). Żeby trzymać się naszego przykładu, wyobraźmy sobie, że Biuro Transportu (*BT*) oferuje dwie możliwości: albo całkowita ochrona przesyłanych informacji przed dostępem do nich innych jednostek, niż te, do których informacja jest skierowana, albo brak takiej ochrony. Użytkownik, czyli Biuro Szyfrów (*BS*) może wybrać jedną z nich przez nadanie odpowiedniej wartości pewnemu parametrowi *BS.SDU*. Poza tym *BS.SDU* może mieć inne parametry oraz zawiera tekst zakodowanej depeszy z dołączonym do niej nagłówkiem *BS* (czyli *BS.PDU*). Inny przykład: Biuro Łączności (*BŁ*), po pewnym okresie działania, zebrało dane statystyczne, na podstawie których ustalono zależność między długością przesyłanych informacji a prawdopodobieństwem błędu

transmisji. W celu utrzymania pewnej założonej efektywności swoich działań, *BŁ* ustaliło nieprzekraczalną długość komunikatu jednorazowo otrzymywanego z *BT*. W tej sytuacji, wydzielona agenda *BT* została zobowiązana do dzielenia komunikatów o długości przekraczającej ten limit na części tworzące pojedyncze *BT.SDU* oraz do składania ich w jedną całość przy odbiorze.

Jak wynika z tych przykładów, między sąsiadującymi warstwami danej jednostki w sieci musi istnieć pewna – zależna od konkretnych rozwiązań i niekiedy skomplikowana – procedura wiążąca te dwie warstwy. Ta procedura nazywa się **sprzęgiem** (ang. *interface*). Sprzęg umożliwia i zapewnia prawidłowe wykorzystanie usług warstwy X dla realizacji protokołu warstwy $X+1$.

NIEZALEŻNOŚĆ WARSTW

W naszym przykładzie znalazła się na końcu opowiastka o zmianach w pracy Biura Łączności zaproponowanych przez szeregowca St. B. Ilustruje ona jedną z ważnych cech architektury warstwowej: możliwości dokonywania wymiany lub modyfikacji warstwy bez konieczności dostosowywania czegokolwiek w innych warstwach. Na ogół jednak takie modyfikacje pociągają konieczność zmian w sprzęgu międzywarstwowym. Inna sprawa, że rozróżnienie między protokołem a sprzęgiem, ważne z metodologicznego punktu widzenia, z wielu powodów nie jest tak jednoznaczne w praktyce realizacyjnej. Dlatego niejednokrotnie niezależność warstwowa bywa nieco pozorna.

LITERATURA

[1] ISO 7498. Information Processing Systems, Open Systems Interconnection. Basic Reference Model, 1984

[2] Mc Quillan J. M., Walden D. C.: The ARPA Network Design Decisions. Computer Networks, Vol. 1, 1977

[3] The Ethernet – A Local Network, Version 1.0. Digital, Intel, Xerox, 1980.

TOMASZ ŻAKOWICZ
Instytut Informacji Naukowej, Technicznej i Ekonomicznej
Warszawa

CDS/ISIS narzędzie do zarządzania bazami danych (1)

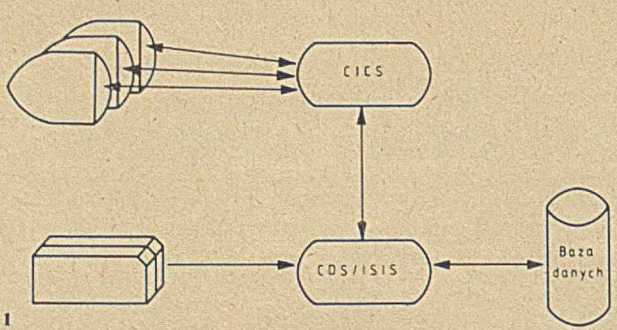
Pakiet programów CDS/ISIS umożliwia tworzenie i obsługę systemów baz danych. W Polsce jest znany od ponad dziesięciu lat. Zdawałoby się zatem, że nie warto o nim pisać, zwłaszcza w dobie rewolucji mikrokomputerowej. Tymczasem systematyczny rozwój pakietu, którego efektem są nowe wersje systemu, spowodował że nawet obecnie oprogramowanie to nic nie straciło ze swej atrakcyjności i „nowości".

W rozwoju CDS/ISIS można wyróżnić dwie charakterystyczne tendencje:
- dostosowanie oprogramowania do nowego sprzętu komputerowego (zgodnego z IBM serii 370), w szczególności do nowoczesnych technik dostępu do plików;
- zwiększenie możliwości funkcjonalnych systemu przez dołączanie nowych podsystemów i narzędzi manipulowania zawartością bazy, a także przez zmianę struktury plików.

Celem artykułu jest przedstawienie podstawowych cech pakietu CDS/ISIS ze szczególnym uwzględnieniem właściwości jego najnowszej wersji o numerze 4.6, rozpowszechnianej od drugiej połowy 1987 roku. Pełną dokumentację pakietu można znaleźć w [2–9].

OGÓLNA CHARAKTERYSTYKA PAKIETU CDS/ISIS

Pakiet CDS/ISIS jest uniwersalnym narzędziem umożliwiającym utworzenie bazy danych zawierających dowolny typ informacji, przy czym zaimplementowane w systemie mechanizmy zapewniają efektywną obsługę głównie baz bibliograficznych. Charakterystyczną cechą pakietu CDS/ISIS jest jego pełna modularność: każdy z programów realizuje ściśle określone funkcje w systemie i może być dostępny niezależnie od pozostałych. Dzięki temu jest możliwe systematyczne doskonalenie pakietu przez wymianę poszczególnych programów na inne, bardziej efektywne. Podstawowe środowisko systemu CDS/ISIS pokazano na rys. 1.



Rys. 1

Użytkownicy dokonują operacji w bazie danych w trybie bezpośrednim (ang. *on-line*) lub wsadowym. Praca w trybie bezpośrednim wymaga pośrednictwa programu CICS (ang. *customer information control system*). Tendencja do odchodzenia od pracy w trybie wsadowym pozwala skoncentrować się na opisie CDS/ISIS z punktu widzenia operatora terminala – zwłaszcza wobec dostępności w języku polskim literatury poświęconej pracy w trybie wsadowym.

W stosunku do początku lat siedemdziesiątych, kiedy dążono do struktury systemu w postaci: *jedna duża baza danych – wielu użytkowników* korzystających jednocześnie z bazy, nastąpiła zmiana podejścia w dwóch kierunkach. Z jednej strony dzięki rewolucji mikrokomputerowej upowszechnił się model: *jedna baza danych – jeden użytkownik*, przy czym zakłada się, że baza ta może być łatwo wymieniona na inną, z drugiej zaś w większych ośrodkach komputerowych upowszechnił się model: *wiele jednocześnie dostępnych baz danych – wielu użytkowników*. Istotną cechą drugiego modelu jest zwykle niezależność geograficzna użytkowników i systemów baz danych realizowana przez utworzenie sieci o dużej przepustowości.

CDS/ISIS, zgodnie z tendencją dominującą w klasie większych komputerów (ang. *mainframe*), jest narzędziem umożliwiającym tworzenie systemów wielu baz danych dostępnych w sieci typu abonenckiego (wielodostęp). W systemie takim jest stosowany jeden plik transakcyjny dla wszystkich baz. Pliki główne i indeksowe baz systemu mogą być grupowane we wspólne klastry (grupy plików) o dostępie VSAM [1] (oddzielne dla plików indeksowych i głównych). Liczba dzienników (tj. plików rejestrowania operacji, ang. *log*) w systemie może być zredukowana do dwóch: dziennika klastra plików głównych i dziennika klastra plików indeksowych (odwróconych). Jednak w takiej sytuacji, biblioteki programów i tablic definicji baz muszą być również wspólne.

W porównaniu z systemami mikrokomputerowymi **zalety** systemów z wieloma bazami danych są następujące:
- duża pojemność bazy danych; teoretycznie bazy danych w systemach mikrokomputerowych mogą być także bardzo pojemne, lecz wprowadzenie do nich dużej liczby rekordów powoduje zmniejszenie efektywności przetwarzania informacji;
- duża pojemność informacyjna systemu, związana z bezpośrednią dostępnością z jednego terminala potencjalnie wielu różnych baz danych;
- większa szybkość przetwarzania; parametr ten zależy oczywiście od stopnia obciążenia systemu i może się zdarzyć, że w systemie nadmiernie obciążonym przetwarzanie jest wolniejsze niż w szybkich systemach mikrokomputerowych;
- możliwość korzystania z wydajniejszych urządzeń zewnętrznych: drukarek, stacji dysków, stacji taśm magnetycznych.

Do **wad** należy zaliczyć:
- uciążliwy dostęp do systemu, związany z istnieniem mechanizmów ochrony danych i koniecznością ochrony kosztownego sprzętu;
- większy koszt obsługi bazy danych, związany z koniecznością kosztownej dzierżawy łączy i innych zasobów mających na ogół bardzo duży udział w kosztach eksploatacji typowego ośrodka przetwarzania danych.

Pakiet CDS/ISIS jest rozpowszechniany na taśmach zawierających następujące biblioteki:
DOCLIB – opisy dokumentacyjne systemu,
SORCLIB – programy w postaci źródłowej,
PARMLIB – karty sterujące do programów usługowych,
MACLIB – makrodefinicje,
PROCLIB – procedury katalogowane,
LINKLIB – programy pakietu w postaci wykonywalnej.
Ostatnia z wymienionych bibliotek jest użyteczna tylko w tych ośrodkach, które dysponują kompilatorem optymalizującym języka PL/I oraz stacjami dyskowymi IBM 3350. W innych ośrodkach bibliotekę tę należy tworzyć korzystając z programów w postaci źródłowej.

[1] VSAM (ang. *Virtual Storage Access Method*) jest techniką dostępu do plików dostosowaną do wirtualnej organizacji pamięci.

Zakłada się, że w ośrodku eksploatacyjnym CDS/ISIS w wersji do numeru 4.4 jest dostępny system operacyjny obsługujący metody dostępu typu VSAM, PAM i SAM, zgodny z systemem OS (np. VS1, VS2). Do obsługi w trybie bezpośrednim jest niezbędny program CICS lub inny zgodny z nimi produkt. Wymagany jest też komputer zgodny z IBM 370, na przykład komputery serii IBM 370, 303*x*, 43*xx*, Amdhal, Intel, Riad. Podstawowe pliki bazy są składowane w pamięciach o bezpośrednim dostępie, typu IBM 2314, 3330, 3340, 3350, 3375, 3380 lub innych zgodnych z nimi. W minimalnej konfiguracji wymaga się jednej stacji dyskowej roboczej i jednej stacji taśmowej (oczywiście niezbędna jest także co najmniej jedna dodatkowa stacja dyskowa na system operacyjny). Należy też dysponować co najmniej jednym czytnikiem kart perforowanych (może być wirtualny) i jedną drukarką. Praca w trybie konwersacyjnym wymaga terminali zgodnych z IBM 3270 lub IBM 2741.

Poszczególne programy pakietu CDS/ISIS są napisane w języku asemblera lub PL/I. Eksploatacja pakietu wymaga więc kompilatora języka PL/I (F lub Optimiser). W trybie wsadowym programy pakietu CDS/ISIS wymagają regionu pamięci operacyjnej o rozmiarze 512 KB. Praca w trybie konwersacyjnym może w zależności od liczby terminali, wymagać obszaru nieco większego. Maksymalna liczba rekordów składowanych w bazie wynosi 999 999 i jest ograniczona sześcioznakowym zapisem numeru rekordu. Maksymalny rozmiar rekordu (w znakach) jest równy wartości parametru CISZ (ang. *control interval size*) pomniejszonej o 33. Domyślnie CISZ = 4096, a zatem bez zmiany tego parametru jest możliwe definiowanie rekordów o maksymalnej długości 4063 znaków. Użytkownicy terminali IBM 3270 mogą definiować pola nie przekraczające 1839 znaków, a inni – 9999 znaków. Liczba pól w rekordzie nie może przekraczać 100, przy czym może być dowolnie dużo wystąpień pól. Musi być spełniony warunek, aby suma długości wszystkich pól i ich wystąpień nie przekraczała limitu dla rozmiaru rekordu pomniejszonego o stałą liczbę 18 znaków.

W celu ułatwienia korzystania z pakietu udostępnia się zbiór 64 procedur katalogowych. Wśród nich są procedury związane z wyszukiwaniem (8 procedur), aktualizowaniem (8), drukowaniem (11), procedury pomocnicze i administracyjne (26) oraz procedury służące do drukowania poszczególnych fragmentów dokumentacji (26). Umowa licencyjna upoważnia użytkownika do modyfikowania tych procedur i (lub) tworzenia nowych, bardziej w jego systemie użytecznych.

Innym rodzajem pomocy dla użytkownika – projektanta aplikacyjnego SZBD, jest język makrodefinicji (ang. CDS/ISIS VSAM *macro language*), pełniący funkcję pomostu między oprogramowaniem CDS/ISIS a programami aplikacyjnymi użytkownika. Język ten ma postać zbioru następujących makrodefinicji, do których można się odwoływać z poziomu asemblera lub PL/I (przez opcję % INCLUDE):
- FILE – deklaracja pliku VSAM,
- GET – odczytanie pliku,
- GETUPD – odczytanie i modyfikowanie pliku,
- PUT – zapisywanie nowych rekordów
- PUTUPD – modyfikowanie lub usuwanie istniejących rekordów,
- POINT – ustawienie wskaźnika plikowego (tryb sekwencyjny),
- ENDFILE – warunek do testowania końca pliku,
- NOTFOUND – warunek do testowania obecności w pliku,
- DUPKEY – warunek do testowania powtórzeń kluczy w pliku,
- NOSPACE – warunek do testowania obecności w pliku miejsca do zapisu rekordu,
- ERROR – warunek do testowania poprawności operacji wejścia-wyjścia,
- CHECK – makrowywołanie testujące wystąpienie błędu w operacji wejścia-wyjścia,
- CLOSE – zamknięcie pliku.

STRUKTURA BAZY DANYCH CDS/ISIS

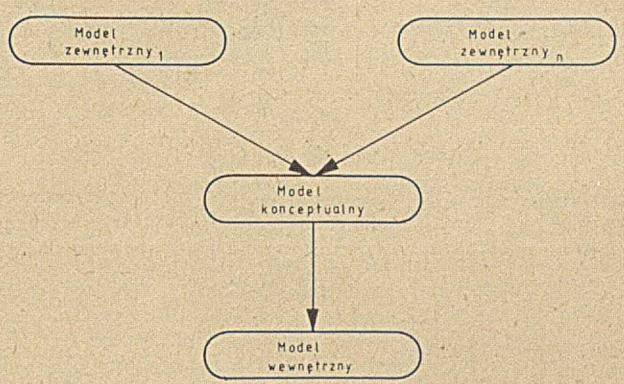
Strukturę bazy danych omówiono na dwóch poziomach: logicznym i fizycznym.

Struktura logiczna

Z punktu widzenia użytkownika, baza danych w systemie CDS/ISIS jest zbiorem *opisów* obiektów. Opis taki składa się ze zbioru wartości uprzednio zdefiniowanych atrybutów obiektu. Dla administratora bazy odpowiednikiem tego modelu jest zbiór rekordów, z których każdy składa się z pewnej liczby pól. W pewnych przypadkach pole rekordu może być w CDS/ISIS sekwencją nierozkładalnych dalej podpól.

Najprostszym modelem bazy danych jest struktura złożona z rekordów o identycznej budowie. Zakłada się, że każde pole w każdym rekordzie zawiera jakąś informację, tzn. nie występują pola nie wypełnione. Baza o takiej postaci może być modelowana za pomocą tablicy, której wiersze odpowiadają rekordom, a kolumny polom w rekordach. W modelu użytkownika wiersze odpowiadają opisom obiektów, a kolumny – ich atrybutom. Nietrudno zauważyć, że taka regularna baza danych jest niewygodna dla szerokiej klasy zastosowań. Często nie ma potrzeby wypełniania wszystkich pól rekordu, zwłaszcza gdy w danej kategorii obiektów zawartość tego pola w żaden sposób tych obiektów nie charakteryzuje. Przykładowo, w bazie danych zawierającej informacje o sportowcach zawartość pola WZROST jest istotna w charakterystyce siatkarza i pole to powinno być wypełnione, w charakterystyce szachisty natomiast informacja o wzroście może być pominięta i pole WZROST pozostanie nie wypełnione.

W wielu *systemach zarządzania bazą danych* (SZBD) są zapewnione programowe możliwości automatycznego wpisywania wartości standardowych do pól podczas wprowadzania rekordów do bazy. Tak jest również w CDS/ISIS. Ponadto użytkownik może pozostawić pole nie wypełnione, pod warunkiem jednak, że pole to jest zdefiniowane jako opcjonalne. Rekord, w którym pole nie zostało wypełnione jest „krótszy" od pełnego zarówno w sensie logicznym, jak i fizycznym. Tak więc rekordy systemu ISIS można uważać za rekordy o zmiennej liczbie pól, przy czym wszystkie możliwe pola rekordu muszą być uprzednio zdefiniowane. Narzuca się zatem strategia projektowania bazy, polegająca na definiowaniu dużej liczby atrybutów, bardzo precyzyjnie charakteryzujących obiekt bazy. Konkretne opisy (wystąpienia rekordów) mogą być wówczas znacznie mniej szczegółowe, dostosowane do ilości dostępnych informacji o obiekcie. Przy takiej strategii jest możliwe uniezależnienie przechowywanych w bazie opisów od sztywnych ram określonych w projekcie bazy.



Rys. 2

W propozycji Zespołu Badawczego Systemów Zarządzania Bazą Danych organizacji ANSI/SPARC [1] wydzielono trzy modelowe poziomy bazy danych: poziom zewnętrzny (nazywany też poziomem użytkownika), poziom konceptualny oraz poziom wewnętrzny (nazywany też poziomem fizycznym). W tej konwencji ogólny schemat bazy danych można przedstawić, jak na rys. 2.

W systemie bazy danych zbudowanym z użyciem pakietu CDS/ISIS model konceptualny bazy jest równoważny pełnej definicji obiektu, a zatem opisowi uwzględniającemu wszystkie atrybuty. Model zewnętrzny odnosi się do definicji częściowej opisu, a więc do zredukowanego zbioru atrybutów. Możliwe jest też zdefiniowanie różnych baz danych odnoszących się do tych samych informacji, lecz inaczej interpretowanych.

Tablica, w której kolumnom odpowiadają atrybuty, a wierszom – opisy obiektów, nie jest wystarczającym modelem dla bazy danych systemu ISIS, gdyż mogą w niej występować rekordy o różnych rozmiarach. Nierówne rozmiary rekordów w bazie wynikają z obligatoryjności wprowadzania informacji tylko do niektórych pól. Okazuje się, że niewystarczające jest również przedstawienie rekordu w postaci „wiersza". Przyczyną tego są chętnie wykorzystywane przez projektantów tzw. *pola powtarzalne*. Pole powtarzalne może zawierać wiele wartości identycznego typu, przy czym liczba tych wartości (wystąpień pola) nie jest zdefiniowana a priori. Dlatego, w ogólnym przypadku na poziomie konceptualnym pojedynczy rekord powinien być widziany nie

jako wiersz o ustalonej liczbie pozycji (struktura liniowa jak w tab. 1), lecz jako pewna dwuwymiarowa struktura (tab. 2).

Tabela 1. Liniowa struktura rekordu

Pole 1	Pole 2	Pole 3	...	Pole N

Tabela 2. Dwuwymiarowa struktura rekordu

Pole 1/wystąpienie 1	Pole 2/wystąpienie 1	...
Pole 1/wystąpienie 2	Pole 2/wystąpienie 2	...
...	...	...

W konsekwencji model bazy danych, będący w najprostszym przypadku regularną tablicą, zostaje uogólniony do nieregularnej bryły trójwymiarowej. W CDS/ISIS występuje więc koncepcja strukturalnie odmiennej bazy danych, dopuszczająca znacznie bogatsze struktury danych niż struktury w systemach relacyjnych baz danych, np. w rozpowszechnionym także w Polsce mikrokomputerowym systemie dBase III.

Elastyczność zdefiniowanych struktur danych może być dalej zwiększona przez „manipulowanie" postacią arkusza wprowadzania do bazy nowych rekordów (ang. *worksheet*) oraz formatem drukowania lub wyświetlania rekordów.

Aczkolwiek do prawidłowego funkcjonowania systemu wystarczy zdefiniowanie jednego formatu obrazowania informacji i jednego arkusza wprowadzania danych, jest możliwe zaprojektowanie wielu takich struktur. Umożliwia to zindywidualizowanie sprzęgów *użytkownik – baza* w zakresie obrazowania informacji, a także zredukowanie pełnego opisu obiektu do opisu pewnego typu zdefiniowanego dla konkretnego użytkownika.

Niech zbiór $T = \{1, ..., n\}$ będzie zbiorem numerów atrybutów obiektów bazy, a P_i zbiorem wartości i-tego atrybutu. *Pełnymi opisami obiektów* nazywamy elementy produktu kartezjańskiego:

$$D = \bigtimes_{i=1}^{n} P_i$$

Dla dowolnego zbioru $L \subset T$ elementy produktu:

$$D^L = \bigtimes_{j \in L} P_i$$

nazywamy opisami typu L.

Typ rekordu bazy jest wyznaczony przez zbiór numerów występujących w nim pól. Może on być określony bądź w chwili wprowadzania rekordu poprzez niewpisanie do niego zawartości niektórych pól rekordu pełnego, bądź też w chwili obrazowania zawartości bazy przez zasłonięcie tych pól, które nie występują w danym typie rekordu. Definiując różne arkusze wprowadzania i formaty obrazowania danych definiuje się tym samym różne typy rekordów „widzianych" przez użytkownika. Jest to równoważne „wycinaniu" z pełnej bazy danych podbaz, zawierających informacje przeznaczone dla poszczególnych użytkowników.

Pełny opis obiektu odwołuje się do modelu konceptualnego bazy, natomiast opis typu L odwołuje się do modelu zewnętrznego bazy.

Innym wymiarem procesu „wycinania" fragmentów bazy danych jest definiowanie przedziałów wyszukiwania w zapytaniach. Wykorzystuje się w tym celu jednoznaczność numerów przypisanych rekordom bazy. Wpowadzając opisy zgodnie z kolejnością ich napływania do systemu, użytkownik zyskuje możliwość podziału zawartości bazy danych na warstwy czasowe. Opcja ta pozwala wyeliminować tzw. szum informacyjny, a także znacznie poprawia efektywność przetwarzania.

Struktura fizyczna

Baza danych w poprzednich wersjach systemu ISIS składała się z pięciu podstawowych plików: głównego – MASTER, transakcyjnego – TRANSACTION, indeksowego – ACCESS, skorowidza pliku głównego – XREF oraz skorowidza pliku indeksowego – INDEX. W obecnej wersji systemu dzięki zastosowaniu metody dostępu VSAM wyeliminowano skorowidze, a wprowadzono pliki dzienników systemowych, oddzielnie dla pliku głównego i systemowego.

W pliku głównym są przechowywane wszystkie składowane w bazie informacje, plik indeksowy natomiast zawiera jednostki wyselekcjonowane z zawartych w bazie opisów wraz z odsyłaczami do rekordów, w których te jednostki są przechowywane. Istnienie pliku indeksowego umożliwia znacznie szybsze wyszukiwanie informacji z bazy danych; taka organizacja jest obecnie implementowana we wszystkich większych SZBD. Plik transakcyjny pełni funkcję bufora informacyjnego. Nowo wprowadzone do bazy informacje lub modyfikacje istniejących rekordów są kierowane do pliku transakcyjnego, gdzie znajdują się do chwili aktualizowania bazy danych. Rekordy zapisane w pliku transakcyjnym nie są widoczne podczas wyszukiwania, jest możliwa natomiast zmiana ich zawartości. Ponieważ proces aktualizacji odbywa się w trybie wsadowym poza sesją użytkowania bazy danych, więc rozwiązanie to nie zapewnia dostatecznie wysokiego stopnia aktualności bazy danych, zwłaszcza w systemie wielodostępnym. Jego zaletą jest natomiast zmniejszenie kosztów aktualizacji bazy i lepsza ochrona jej zawartości.

W nowej wersji CDS/ISIS (wersja 4.6) użytkownik ma możliwość bezpośredniego zapisu do pliku głównego w trybie wsadowym. Rola pliku transakcyjnego jako bufora informacyjnego została zachowana jedynie w odniesieniu do trybu konwersacyjnego, ze względu na trudności w rozwiązywaniu kolizji spowodowanych – na przykład – chęcią jednoczesnego odczytania i zapisywania tego samego rekordu przez różnych użytkowników. Aktualizacja bazy danych jest dokonywana okresowo (według zaleceń – pod koniec dnia) na podstawie pliku transakcyjnego. W tym czasie baza jest blokowana dla wszystkich operacji.

Pliki skorowidzowe poprzednich wersji systemu umożliwiały szybkie wyszukiwanie, lecz operacja aktualizacji była znacznie bardziej kłopotliwa, gdyż przy dokonywaniu zmian wymagała zachowania zgodności w czterech powiązanych ze sobą plikach. Organizacja plików VSAM umożliwia szybkie wyszukiwanie i oszczędną gospodarkę pamięcią, dzięki czemu można zrezygnować z dotychczasowej, nader „kunsztownej" organizacji plików bazy. Dużemu uproszczeniu uległ tym samym proces aktualizacji i wzrosła niezawodność systemu. Dodatkowym środkiem zwiększającym niezawodność jest zastosowanie dzienników (dla plików głównego i indeksowego), w których są rejestrowane wszelkie zmiany zawartości bazy.

PROJEKTOWANIE BAZY DANYCH

Baza danych w CDS/ISIS zawiera rekordy o identycznej strukturze. Proces projektowania bazy sprowadza się zatem do zdefiniowania poszczególnych pól rekordu przez utworzenie tablicy FDT (ang. *field definition table*), a także do zdefiniowania środowiska programowego. Projekt środowiska programowego obejmuje adaptację procedur katalogowych do konkretnej instalacji sprzętowej oraz podanie zestawu definicji: arkuszy wprowadzania danych, wzorców drukowania, sposobu indeksowania, reguł sortowania wyszukanych w bazie rekordów, zakresu wyszukiwania rekordów, a także zbioru standardowych zapytań do systemu. Niektóre z tych definicji są zawarte w tablicy FDT, a inne są grupowane w zestawy nazywane tablicami FST (ang. *field select table*).

Tablice FDT i FST

Tablica FDT jest sekwencją 80-kolumnowych kart. Składają się na nią:
- jedna lub dwie karty tytułowe,
- karty komunikatów wyświetlanych przy autoryzacji użytkowników;
- karty autoryzacyjne;
- karty opisu pól rekordów bazy danych,
- karta ⓐ,
- karta formatu weryfikacji poprawności wprowadzanych danych (opcjonalnie);
- karty definiujące (dla trybu bezpośredniego): arkusze wprowadzania danych, formaty wydruku kontrolnego wprowadzonych rekordów, formaty wydruku (wyświetlania) wyszukanych rekordów;
- zbiór kart zawierających zadania w języku JCL dla operacji przekazywania zapytań do wsadowego wykonania w trybie bezpośrednim.

Wiele z wymienionych kart definiuje ściśle techniczne aspekty pracy w systemie CDS/ISIS, np. definicje arkuszy wprowadzania danych lub

obrazowania wyszukanych w bazie rekordów. Omówienia wymagają te z kart, które wskazują na specyficzne właściwości **systemów baz danych**[2] utworzonych w środowisku CDS/ISIS.

W karcie tytułowej, oprócz identyfikowania bazy danych i hasła chroniącego przed wprowadzaniem danych w trybie bezpośrednim przez osobę nieupoważnioną, znajdują się również opcjonalnie wypełniane obszary zwane *szczelinami* (ang. *slots*). W szczelinach są zapisywane nazwy plików systemowych służących do weryfikacji poprawności indeksowania, a także nazwy modułów utworzonych przez użytkownika, których zadaniem jest weryfikacja poprawności terminów wprowadzanych do bazy danych lub tworzenie wybranych pól. Do weryfikacji poprawności indeksowania można również użyć słownika (wchodzącego w skład pliku indeksowego) innej bazy danych. Powiązanie takie definiuje się w drugiej karcie tytułowej.

W karcie ⓐ są zdefiniowane inne połączenia definiowanej bazy danych z systemem baz danych. W szczególności jest w niej podany opcjonalnie identyfikator tezaurusa sprzężonego z definiowaną bazą oraz – gdy dla danej bazy „fizycznej" zaprojektowano więcej tablic FDT – nazwa „fizycznego" zbioru danych, z jakim jest sprzężona baza, definiowana w danej tablicy FDT.

W kartach definicji pól są zawarte takie informacje, jak: identyfikator pola, status pola, maksymalny rozmiar w znakach, sposób ewentualnej weryfikacji poprawności wprowadzanych do tego pola danych oraz kod typu definiowanego pola, związany z techniką tworzenia wartości indeksowej z jego zawartości.

Tablice FST oprócz specyfikacji sposobu tworzenia wartości indeksowych umożliwiają definiowanie *pełnego zapytania* do systemu. Zapytanie takie zawiera: formuły wyszukiwawcze, specyfikację zakresu przeszukiwania bazy danych, specyfikację formatu wydruku i sortowania rekordów przygotowywanych do obrazowania. Z takich zdefiniowanych pierwotnie zapytań może korzystać użytkownik, np. przy periodycznym tworzeniu katalogów nowych nabywców biblioteki. Definicje zapytania i wartości indeksowych stanowią elementy jednej biblioteki tablic FST.

Indeksowanie

W CDS/ISIS istnieje jeden globalny plik indeksowy. Elementami tego pliku, tj. wartościami indeksowymi, mogą być bądź indywidualne terminy lub kody, bądź też frazy. Poprawność selekcjonowanych indeksów może być weryfikowana według techniki systemowej lub innej implementowanej przez użytkownika. W pierwszym wariancie wybierane indeksy są porównywane z terminami występującymi w specjalnym indeksowo-sekwencyjnym (a w najnowszej wersji CDS/ISIS z dostępem VSAM) pliku LOOKUP, bądź w pliku indeksowym innej bazy danych systemu. Kontrola poprawności indeksowania na podstawie pliku indeksowego jednej, wzorcowej bazy systemu umożliwia ujednolicenie słownictwa stosowanego w całym systemie.

Na podstawie pliku indeksowego są tworzone wyrażenia wyszukiwawcze umożliwiające znalezienie tych dokumentów, w których wyspecyfikowany indeks występuje. W przeciwieństwie do systemu STAIRS, w CDS/ISIS nie są tworzone profile dokumentów, nie ma procedur klasteryzacyjnych i statystycznych technik analizy zgodności opisów dokumentów z zapytaniem. Uzyskanie dużej efektywności wyszukiwania w CDS/ISIS wymaga zarówno precyzyjnego indeksowania, jak i precyzyjnego formułowania zapytań wyszukiwawczych.

Sposób wybierania z zawartości pola wartości indeksowych jest zdefiniowany w jednej z tablic FST. Ponieważ w systemie istnieje jeden globalny plik indeksowy, dogodnie jest specyfikować w tablicy FST tzw. *prefiksy indeksów*, identyfikujące pole, z którego została wybrana dana wartość indeksowa. Dzięki temu ulega uproszczeniu krytyczny w CDS/ISIS proces kontrolowania zawartości pliku indeksowego.

W pewnych sytuacjach zachodzi potrzeba utworzenia słownika do weryfikacji poprawności wartości indeksowych (*LOOKUP file*) i (lub) *pliku grup indeksowych (ANY file)*. Grupa indeksowa składa się z pewnej liczby wartości indeksowych, którym jest przypisany identyfikator. Użycie tego identyfikatora w zapytaniu powoduje zastąpienie go alternatywą elementów grupy indeksowej. Przykładowo, identyfikator IMIE będzie każdorazowo zastępowany wyrażeniem:

AGNIESZKA OR BEATA OR CEZARY OR HENRYK

o ile imiona te są zawarte w grupie indeksowej IMIE i nie są w niej zawarte żadne inne wartości indeksowe.

Formaty wydruku i wyświetlania

Projektant systemu bazy danych w CDS/ISIS ma dużą dowolność w kształtowaniu postaci informacji wychodzących z systemu. Korzysta on w tym celu ze specjalnego języka PFL (ang. *print formatting language*), umożliwiającego tworzenie formatów wydruku (wyświetlania). Dla strony wydruku (lub obrazu na ekranie) można definiować wielkości marginesów, rozmiary przerw między wydrukami poszczególnych pól, tryb prawostronnej justyfikacji, a także, wiele elementów warunkowych, drukowanych (wyświetlanych) w zależności od tego czy konkretne pole lub podpole jest wypełnione albo czy pole występuje. Warunkowo może się odbywać m.in. drukowanie zawartości poszczególnych pól, wstawianie lub zmienianie odstępów między wydrukiem kolejnych pól oraz umieszczanie stałych tekstowych towarzyszących określonym polom. Problemem charakterystycznym dla wszystkich rozwiązań CDS/ISIS jest nadmierna „sztywność" języka PFL, który wymaga bardzo uważnego kodowania wyrażeń.

Formaty wydruku są składowane w bibliotece tablic FST, a także, o ile dotyczą trybu bezpośredniego, stanowią elementy tablicy FDT. W szczególności musi być zaprojektowany jeden format wydruku kopii dokumentacyjnych rekordów wprowadzanych w trybie bezpośrednim, a także co najmniej jeden format wydruku dokumentów znalezionych w wyniku wyszukiwania w tym trybie. Aby uzyskać większą wygodę przy wprowadzaniu informacji do bazy należy również zdefiniować co najmniej jeden *arkusz wprowadzania*. Arkusz ten definiuje układ graficzny ekranu, kolejność wypełniania, wartości domyślne i status obowiązkowości wprowadzania informacji dla poszczególnych pól. Zapamiętany w bibliotece format wydruku lub arkusz wprowadzania może być w dowolnej chwili użyty lub zastąpiony innym.

W CDS/ISIS można także zaprojektować *sekwencje sortowań*. Na ogół użytkownik jest zainteresowany tym, aby informacje wychodzące z systemu były uporządkowane. CDS/ISIS umożliwia dość precyzyjne sortowanie według dowolnego jednego pola rekordu lub według łącznej zawartości zestawu pól. Możliwe jest także sortowanie wielostopniowe (maksymalnie cztery poziomy). Niezbędne jest wówczas ustalenie sekwencji sortowań, a więc określenie pola (zestawów pól), według którego mają odbywać się kolejne sortowania. Każde następne sortowanie powoduje uporządkowanie grup rekordów o identycznej zawartości pola „sterującego" poprzednim sortowaniem.

LITERATURA

[1] ANSI/X3/SPARG Study Group on Data Base Management Systems. Bulletin of ACM SIGMOD, 7, No. 2, 1975
[2] CDS/ISIS Catalogued Procedures. UNESCO, 1987
[3] CDS/ISIS Data Manipulation Language Manual Documentation. UNESCO, 1987
[4] CDS/ISIS General Description Documentation. UNESCO, 1987
[5] CDS/ISIS Library Acquisition. UNESCO, 1987
[6] CDS/ISIS Photocomposition Documentation. UNESCO, 1987
[7] CDS/ISIS System Installation Documentation. UNESCO, 1987
[8] CDS/ISIS Terminal Operator Manual Documentation. UNESCO, 1987
[9] CDS/ISIS Thesaurus Subsystem Manual Documentation. UNESCO, 1987

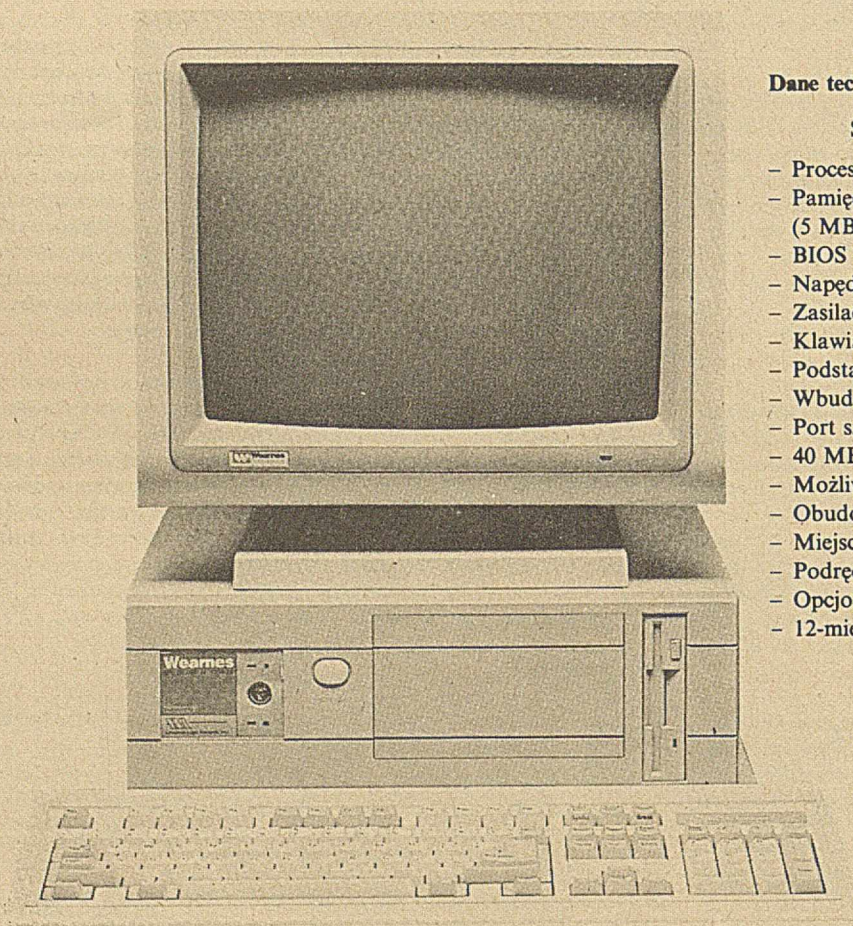
[2] W nomenklaturze CDS/ISIS systemem baz danych nazywa się zestaw różnych baz (każda z rekordami jednego typu) przetwarzanych w tym samym środowisku sprzętowym i programowym.

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 – 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40-045 Katowice
telefon i telefaks: 518–628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. M. Curie Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 25325

MARIA BRONOWSKA
Instytut Podstaw Informatyki PAN
Centrum Obliczeniowe
Warszawa

System graficzny Ramtek RM-9460

W połowie 1988 r. w Centrum Obliczeniowym IPI PAN zainstalowano minikomputer dużej mocy Delta-8000, typu VAX/750 wyposażony m.in. w system graficzny RM-9460 Ramtek, jedno z najbardziej wydajnych urządzeń graficznych w kraju. System ten jest wieloprocesorowym urządzeniem graficznym, które dekoduje i przetwarza zbiór rozkazów wysokiego poziomu, wysyłanych z komputera centralnego. Każdy rozkaz składa się z jednego lub kilku 16-bitowych słów. Odebrane rozkazy mogą być bezpośrednio wykonywane lub zapamiętywane (do późniejszego wykonania) w specjalnej pamięci użytkownika w komputerze centralnym lub w module procesora systemowego Ramtek, jako fragmenty obrazów (tzw. listy obrazowe). Rozkazy bezpośrednio lub pośrednio adresują w układzie współrzędnych globalnych (współrzędnych obrazu) przestrzeń o rozmiarze 32 K × 32 K jednostek. System jest wyposażony w bloki funkcjonalne, wykonujące operacje skalowania obrazu oraz jego obcinania do zadanych granic, równoległych do brzegu ekranu. Powiększenie lub zmniejszenie rysunku osiąga się przez transformację współrzędnych bez zmiany cech elementów obrazu: grubości i wzoru linii oraz rozmiaru znaków. Rysunek jest zapisywany w pamięci obrazu przez ponowne wykonanie listy obrazowej. Wizualizacja jest dokonywana sprzętowo z możliwością wykonywania panoramy dynamicznej i zbliżenia (ang. *panning* i *zooming*), pod kontrolą programu synchroniującego (ang. frame-synchronized program).

Ekran monitora kolorowego, stanowiącego wyposażenie systemu Ramtek, ma rozmiar 1280 × 1024 jednostek. Równoczesna praca wielu układów przetwarzających zapewnia znaczne szybkości generowania obrazu rastrowego na podstawie listy obrazowej (tj. konwersji struktury obrazu), do 16 000 wektorów/s lub 892 000 pikseli/s. Szybkość generowania znaków zależy od ich wielkości i wynosi maksymalnie 2800 znaków/s.

BUDOWA SYSTEMU

Połączenie między komputerem centralnym a systemem RM-9460 zapewnia 16-bitowy uniwersalny sprzęg równoległy (GPIF), umieszczony w procesorze systemowym Ramteka. W module tego procesora są również umieszczone trzy łącza szeregowe, zegar, kanał DMA i układy przerwań. Dekodowanie rozkazów kanałowych, zapisywanie danych graficznych do pamięci użytkownika (list obrazowych i krojów znaków), transformacja geometryczna na danych graficznych oraz uruchamianie procesora graficznego są wykonywane przez procesor systemowy. Moduł tego procesora wykonano z użyciem układu scalonego MC 68000 oraz pamięci 128 KB EPROM i 256 KB RAM (z których 240 KB stanowi pamięć użytkownika).

Procesor graficzny przetwarza listy obrazowe z pamięci użytkownika, generując pierwotniki (alfanumeryczne, graficzne, odwzorowania bitowe itd.), i zapisuje je do pamięci obrazu. Wykonuje także obcinanie, wykrywanie elementów (podobrazów), panoramę dynamiczną i zbliżenie, wykorzystując w tym celu układy generowania adresów odświeżania oraz 16-bitowy procesor bipolarny specjalnego przeznaczenia, w którego skład wchodzą cztery jednostki arytmetyczno-logiczne (ALU) z odpowiednią pamięcią sterującą (ROM) i rejestry (RAM). Przeciętny czas wykonania jednego rozkazu wynosi 284 ns.

Pamięć obrazu pozwala zapamiętać obraz w postaci matrycy punktów 1280 × 1024, przy czym jeden piksel jest opisany za pomocą 16-bitowego słowa. Przeciętny czas dostępu wynosi 1,49 μs.

Generator wideo rozwija obraz, tzn. przekształca informację zapamiętaną w pamięci obrazu na sygnały odpowiednie dla monitora Ramteka i ewentualnie innych monitorów, projektorów o dużych ekranach lub drukarek. Rozwijanie obrazu odbywa się na zasadzie odczytywania zawartości pamięci obrazu dla kolejnych pikseli, dekodowania odczytanych danych zgodnie z tablicą pośrednią koloru (wczytaną wcześniej do systemu Ramtek i zapamiętaną w pamięci PROM/RAM) i dołączania koloru (intensywności dla monitora monochromatycznego) do odpowiednich sygnałów sterujących kolorami podstawowymi lub poziomem szarości.

Klawiatura alfanumeryczna i graficzne urządzenie wejściowe są sprzężone z systemem Ramtek za pomocą łączy szeregowych. Moduł tych łączy zawiera cztery porty klawiaturowe, cztery porty sterowania kursorem, cztery generatory kursora oraz programy obsługi: klawiatury, kuli, pióra świetlnego i czytnika rysunków.

OPIS DZIAŁANIA

Listy obrazowe mogą być pamiętane albo w komputerze centralnym i realizowane w czasie rzeczywistym, albo w pamięci list obrazowych, znajdujących się w systemie Ramtek. W procesorze systemowym Ramtek zarezerwowano 240 KB na listy obrazowe. Są one ciągami rozkazów graficznych, definiujących rysunek we współrzędnych bezwzględnych. Listy obrazowe z pamięci obrazów mogą być wywoływane jako podprogramy (podobrazy) również przez komputer centralny, dołączane do kursora w sterowniku kursora lub do klawisza na klawiaturze. Maksymalna długość podprogramu wynosi 16 KB. Podprogram może być jednak wywoływany z innego podprogramu, dzięki czemu można wydłużyć listę obrazową i wykonywać większe rysunki. Listy mogą być również modyfikowane przez komputer centralny. Można zmieniać parametry obrazu: kolor, rodzaj linii, skalę, kąt obrotu itp. Dane zawarte w listach obrazowych są definiowane we współrzędnych użytkownika. Są one automatycznie transformowane na współrzędne globalne podczas przetwarzania listy obrazowej i zapisywania tych danych do pamięci obrazu. Rysunek lub segment rysunku może być skalowany i obracany bez zmiany listy obrazowej opisującej obiekty. Ponieważ przestrzeń globalna jest znacznie większa niż przestrzeń pamięci obrazu, więc można wyświetlać również fragmenty bardzo dużych lub szczegółowych rysunków.

Efekt panoramy dynamicznej, tj. bocznego przemieszczania obrazu na ekranie, jest tworzony przez ustalenie położenia pamięci obrazu względem współrzędnych obrazu, czyli ustalenie położenia okna w przestrzeni globalnej i powtórzenie procesu przetwarzania listy obrazowej. Zapisany w pamięci obrazu rysunek może być jeszcze modyfikowany za pomocą sprzętowo realizowanego przesuwania i powiększania.

Bardzo przydatną, chociaż rzadko spotykaną w systemach graficznych funkcją jest tzw. porządkowanie. Uzależnia ono widoczność graficznej informacji zawartej w rozkazie od wartości argumentu „klasa wyświetlania" tego rozkazu. Jeżeli wartość argumentu leży wewnątrz zakresów ustalonych przez rozkaz zewnętrzny w stosunku do listy obrazowej, to informacja jest wyświetlana na ekranie.

Zdolność wykrywania podobiektów umożliwia m.in. identyfikowanie wektora lub innych elementów graficznych występujących w liście obrazowej, przez wskazanie dowolnej jego części. W komputerze wektor jest wyznaczony przez współrzędne końców w przestrzeni globalnej i jego identyfikowanie wymaga dużych nakładów obliczeniowych. Cecha wykrywania identyfikuje podprogramy i rozkazy graficzne, które wpisują informację wewnątrz podanego obszaru. Obszar ten jest ustalonym prostokątem w przestrzeni współrzędnych globalnych i jest on parametrem rozkazu wykrywania. Zostaje ustalony w położeniu kursora, ustawianego przy użyciu kuli, drążka lub pióra świetlnego. Po wskazaniu obiektu i przesłaniu przerwania do komputera jest ustawiany tryb umożliwiający wykrywanie i zostaje wywołana retransmisja listy obrazowej z pamięci użytkownika. Lista jest wykonywana bez zapisu informacji do pamięci obrazu. Dla rozkazów próbujących zapisywać zostają zapamiętane: identyfikator podprogramu, klasa wyświetlania i adres rozkazu w liście obrazowej; informacje te są dostępne dla

komputera. Możliwe jest wykorzystanie detekcji całych podprogramów, np. komputer jest informowany o wskazaniu elementu spisu funkcyjnego lub podzespołu na rysunku technicznym.

Do struktury organizacyjnej systemu wprowadzono pojęcie jednostki kontekstu. Jednostka kontekstu jest pewną strukturą danych opisującą system, w którym działa użytkownik – określa tzw. środowisko graficzne, tzn. zasoby użytkownika oraz zmienne wymagane w jego procesie wyświetlania. Gdy użytkownik jest czasowo nieaktywny, nie są potrzebne żadne dane zewnętrzne ani nie ma potrzeby zachowania żadnego specjalnego stanu. Użytkownik może mieć kilka jednostek kontekstu do różnych celów, np. kanał wyświetlania i kanał tekstowy lub może dzielić je z innymi użytkownikami.

Ważną cechą jednostek kontekstu jest włączanie wielu grup pamięci jako miejsca przeznaczenia danych. Wszystkie wybrane grupy pracują z tymi samymi parametrami: punktem bieżącym, oknem itd. Ponieważ przekazywane dane są takie same, wszystkie obrazy są identyczne.

Parametry kontekstu można podzielić na trzy sekcje:
Sekcja parametrów procesora graficznego zawiera listę grup pamięci w każdym procesorze, do których można zapisywać dane, listę procesorów z zezwoleniem zapisu oraz listę parametrów opisujących miejsce i sposób generowania danych. Lista parametrów zawiera: punkt bieżący, wartości koloru i tła oraz wszystkie parametry opisujące stan zmiennych sprzętowych.
Sekcja parametrów programowych jednostki kontekstu zawiera wszystkie parametry potrzebne do wykonania rozkazów wyświetlania. Parametry te określają dane, które są tworzone oraz sposób wykonania rozkazów. Są tu włączone wszystkie zmienne rozkazów, wektory podprogramów przetwarzania i wskaźniki do używanego kroju znaków.
Sekcja krojów znaków jednostek kontekstu zawiera parametry tekstu (np. rozmiar) i krój znaków. Nie jest ona bezpośrednio zawarta w bloku kontekstu. Do niej odwołuje się sekcja parametrów programowych. Podczas wczytywania nowego kontekstu jest zastępowany wskaźnik, a bloki tekstu są również dostępne przez odwzorowanie.

PODSTAWOWE ELEMENTY LISTY OBRAZOWEJ

Lista obrazowa może być tworzona z następujących elementów:
- pierwotniki graficzne (ang. *primitives*) – punkt, wypełniony prostokąt, wektor, krzywa drugiego stopnia, okrąg, łuk, wypełniony wielokąt, tekst, symbol specjalny, wykres, wykres słupkowy;
- atrybuty – kolor (intensywność) przedniego planu i tła, rodzaj linii, rozmiar i orientacja znaku, okno, rewers tła, nadpisywanie (ang. *additive write*);
- kroje znaków – krój standardowy tj. 64 znaki w matrycy 7×9 jednostek, dodatkowo 17 zbiorów po 128 znaków; maksymalny rozmiar znaku wynosi 16×20 jednostek, dalsze powiększanie jest możliwe przez powtarzanie pikseli;
- transformacje współrzędnych – przesunięcie (w dwóch wymiarach) obrót, skalowanie;
- tryby adresowania – bezwzględny, względny, indeksowy: przewidziano oddzielne rozkazy przeniesienia współrzędnych kursora do pożycji pióra (COP) lub rejestrów indeksu;
- rozmiar obszaru – $32K \times 32K$ jednostek (rozmiar przestrzeni globalnej);
- rozmiar ekranu – 1024×1289 jednostek;
- funkcje wideo – panorama dynamiczna i zbliżenie (w całkowitych krokach między 1:1 i 1:16), miganie, przejście od koloru do poziomu intensywności;
- fragmenty rysunku – zapamiętywane lokalnie jako podprogramy graficzne w segmentach pamięci 4–16 KB,
- porządkowanie – kontrola widoczności podprogramów i rozkazów graficznych;
- wykrywanie występowania – identyfikowanie podprogramów i rozkazów generujących dane w podanym prostokątnym oknie;
- obcinanie – ograniczenie rysunków do określonego obszaru ekranu;
- okno formatu – rozpoczęcie nowego wiersza lub strony, gdy bieżąca jest zapełniona do końca;
- obrót – obrót elementów graficznych co 1°, a kroju znaków, siatki rastrowej, rysunku co 90°;
- przeglądanie – w górę, w dół, w lewo, w prawo; cały ekran lub wewnątrz określonego prostokątnego obszaru;
- odczytywanie – odczytanie zawartości pamięci użytkownika, pamięci obrazu, tablic pośrednich kolorów, współrzędnych kursora oraz informacji o stanie;
- przerwania – powodowane przez aktywność urządzeń zewnętrznych i niedozwolone rozkazy,
- opóźnienie – synchronizowanie wykonania rozkazu graficznego z impulsem pionowym lub odnawianiem określonej linii rastrowej.

FORMAT I LISTA ROZKAZÓW

Rozkazy systemu Ramtek można podzielić na trzy grupy. **Pierwsza grupa** zawiera rozkazy (normalnego formatu), wpisujące dane do pamięci obrazu, a przez nią na ekran. Rozkazy te są poleceniami poziomu makro, ukierunkowanymi na zastosowanie. Ich funkcje zmieniają się zależnie od argumentów dostarczanych jako argumenty wywołań pakietu graficznego lub programu obsługi wejścia-wyjścia znajdujących się w komputerze. Argumenty umożliwiają wybór kanału, kolor lub intensywność, skalę, rodzaj linii itd. Rozkazy z **drugiej grupy** są ukierunkowane na sprzęt. Mają format specjalny, który zmienia się zależnie od wykonywanej funkcji. Wykonują wczytywanie krojów znaków i tablic pośrednich koloru. **Trzecia grupa** zawiera rozkazy definiowane przez użytkownika, który wczytuje własny program sterujący, interpretujący te rozkazy. Są to, na przykład, rozkazy bezpośrednio interpretujące bazę danych komputera lub wykonujące funkcje wysokiego poziomu ukierunkowane na zastosowanie.

Rozkazy o formacie normalnym są bardzo elastyczne. Argumenty i dane są opcjonalne; wiele rozkazów zawiera jedno lub dwa słowa:
- słowo kontrolne – pierwsze słowo zawierające kod operacji i bity kontrolne definiujące funkcję i (lub) format rozkazu:
IX – wybór trybu adresowania (bezwzględne, względne oraz dwuadresowe),
AD – tryb pisania (zastępowanie lub nadpisywanie) znaków, pola rastrowego, danych graficznych,
BK – zamiana miejsca koloru i tła,
RP – odwracanie kolejności odpakowywania i przetwarzania danych bajtowych,
DF – wystąpienie lub brak słowa liczby bajtów danych;
- słowo liczby bajtów danych – zawiera liczbę bajtów danych przekazywanych do lub z systemu RM-9460;
- dane – składają się z bajtów, słowa lub wielu słów, zależnie od funkcji wykonywanej przez system RM-9460;
- słowa wskaźników argumentów – pokazują stan 22 argumentów; pierwsze sygnalizuje 16 pierwszych, a drugie – 6 argumentów;
- argumenty – są różnymi wewnętrznymi wielkościami, jak maska koloru, tło, okno formatu, początek układu współrzędnych ekranu itp., określającymi operacje dla wszystkich rozkazów o normalnym formacie; każdy zbiór lub podzbiór argumentów jest wyznaczony przez stan 22 bitów w słowach wskaźnikowych argumentów.

Lista rozkazów zawiera 148 rozkazów, przy czym 25 z nich to rozkazy typowo graficzne, m.in. rysowanie punktów, wektorów, łamanej, okręgu i łuków okręgów, krzywej drugiego stopnia, prostokątów lub wielokątów wypełnionych dowolnym kolorem, różnego rodzaju wykresów, napisów tekstowych itp.; 21 rozkazów dotyczy funkcji typowo arytmetycznych i logicznych np. tworzenie macierzy transformacji do skalowania obrotu i przesunięcia, mnożenie jej przez liczbę lub inną macierz, układanie na stos i zdejmowanie ze stosu; 16 rozkazów służy do zarządzania listami obrazowymi, wśród nich m.in. przydzielanie i zwalnianie pamięci dla list, wczytywanie i wywoływanie list obrazowych, ustawianie, dodawanie, odejmowanie oraz operacje logiczne na 8 rejestrach list obrazowych. Dalszych 27 rozkazów umożliwia posługiwanie się urządzeniami wejściowymi i sterowanie kursorem, np. czytanie i ustawianie położenia kursora we współrzędnych lokalnych, globalnych, pamięci obrazu i ekranu, odczytanie stanu urządzenia zewnętrznego, powiązanie urządzenia z kursorem, ustawianie stanu echa klawiatury; 8 rozkazów dotyczy wykrywania podobrazów: początek i koniec procesu wykrywania, odczytywania bufora zawierającego rezultaty wykrywania itp.; 6 rozkazów umożliwia wprowadzanie programowanego przez użytkownika kroju znaków, 5 daje możliwość tworzenia, zapamiętywania, odczytywania i odtwarzania kontekstów. Pozostałe rozkazy spełniają zadania pomocnicze, np. ustawianie normalnych parametrów, odczytywanie parametrów, odczytywanie konfiguracji, wybór położenia początku układu współrzędnych (lewy górny lub dolny róg ekranu).

LITERATURA

[1] RM-9460 Graphic Display System Software Reference Manual. Ramtek Corp, No. 8000081-02A, July 1983
[2] VAX/VMS RM-9460/RM-9460 Device Driver. Ramtek Corp., No. 8000085-01C, April 1985.

HENRYK J. RUNKA
Instytut Cybernetyki Ekonomicznej
Akademia Ekonomiczna
Poznań

Duża macierz w Turbo Pascalu 5.0

We wszystkich obecnie stosowanych algorytmicznych językach programowania komputerów, implementację wektora można przeprowadzić za pomocą tablicy jednowymiarowej, natomiast implementację macierzy za pomocą tablicy dwuwymiarowej. W języku Pascal, definiowany pierwotnie typ strukturalny *array* jest wbudowanym narzędziem implementacji tych obiektów. Jednakże stałe ich rozmiary w programach, niezależnie od rozmiaru zadań problemu, powodują że ta implementacja nie zawsze jest efektywna. W Turbo Pascalu 5.0 obowiązuje ponadto ograniczenie rozmiaru pamięci przydzielanej obiektowi, co zostało przedstawione szczegółowo w [2]. Korzystając z wolnej pamięci komputera (a nie obszaru danych programu) można w Turbo Pascalu 5.0 dynamicznie przydzielać pamięć wektorowi, jeśli stosuje się wskaźnik (zmienną typu wskaźnikowego) na obiekt typu strukturalnego *array*. Pamięć należy jednak przydzielać za pomocą procedury GetMem lub też funkcji MyGetMem przedstawionej w [2]. Rozmiar przydzielonej obiektowi pamięci może być wtedy taki, jaki wynika z rozmiaru zadania problemu. Przykładem tego podejścia może być następujący program przedstawiony na wydruku 1.

```
program LargeString;

uses PackedString, WindStack, CRT;

const
      PackedStrLen  = 65520;

type
      charPackedArray          = packed array[ 0..PackedStrLen - 1 ]
                                 of char;
      Ptr_to_charPackedArray   = ^charPackedArray;


function XCopy(  p     : pointer;
                 f,
                 count : word
                 : string;

var
    i, Xlen  : word;
    strg     : string;

begin
  Xlen := XLength( p );
  if f > Xlen - 1 then begin
    Xcopy := '';
    Exit;
  end;
  if f + count > Xlen then
    count := Xlen - f;
  strg := '';
  for i := f to f + count - 1 do
    strg := strg +  Ptr_to_charPackedArray( p )^[ i ];
  Xcopy := strg;
end;

var
    MyString : pointer;
    n        : word;

begin
  writeln;
  write( 'How many characters? ' ); readln( n );
  if n = 0 then n := 1;
  MyString := ALLOC_STRING( n );
  XInsert( 'Turbo Pascal 5.0', MyString, 0 );
  writeln( XCopy( MyString, 0, 12 ) );
  Wait_to_Go;
end.
```

Wydruk 1. Dynamiczna alokacja upakowanego ciągu znaków

DYNAMICZNA ALOKACJA UPAKOWANEGO CIĄGU ZNAKÓW

Funkcja XCopy dostarcza napisu (wartość typu *string*) z innego upakowanego ciągu znaków (wartość typu *charPackedArray*). Funkcje XLength i ALLOC_STRING oraz procedura XInsert są zdefiniowane w jednostce PackedString. Upakowany ciąg znaków jest zakończony znakiem # 0 (NULL), wstawianym przez funkcje ALLOC_STRING. Przyjęta definicja upakowanego ciągu znaków jest zgodna z definicją ciągu znaków w języku C (char*MyString). Maksymalna długość upakowanego ciągu znaków wynosi 65 519 znaków oraz znak zakończenia # 0, natomiast wielkość przydzielonej pamięci przez funkcję ALLOC_STRING jest wielokrotnością liczby 16 i należy ją dostosować do rozmiaru danych.

To samo podejście w przypadku implementacji macierzy za pomocą tablicy dwuwymiarowej nie jest już tak efektywne. Zatem stosując definicje i deklaracje:

```
const
    maxRow = 50;     maxCol = 60;
type
    ElementType      = real;
    vector           = array[1..maxCol] of ElementType;
    matrix           = array[1..maxRow] of vector;
    ptr_to_matrix    = ^matrix;
    ptr_to_vector    = ^vector;
var
    A : pointer;
```

można przydzielić zmiennej wskaźnikowej *A* dynamicznie pamięć. Jednakże przy rozmiarze macierzy $m \leqslant$ maxRow, przydzielony obszar musi mieć długość przynajmniej m*maxCol*SizeOf (ElementType) bajtów. Pamięć tę należy również przydzielić za pomocą procedury GetMem lub funkcji MyGetMem. Zapotrzebowanie na pamięć zależy jednak od stałej maxCol, dlatego rozmiar pamięci jest zredukowany tylko o ostatnie niewykorzystane wiersze macierzy. Macierz jest bowiem zapamiętana wierszami i dla prawidłowego wyznaczenia pozycji elementu, każdy wiersz musi zawierać *maxCol* elementów. Przykładowo:

```
ptr_to_matrix (A)^[2][1]
```

jest (maxCol + 1)-szym elementem obszaru macierzy.
Ażeby więc uniezależnić rozmiar przydzielonej pamięci macierzy $A_{n \times m}$ od stałych *maxCol* i *maxRow* należy zastosować inne implementacje.

IMPLEMENTACJA MACIERZY ZA POMOCĄ WEKTORA WSKAŹNIKÓW NA WIERSZE

Stosując tę implementację nie wychodzi się poza definicje języka Turbo Pascal 5.0 (nie występują operacje na wskaźnikach, które zaproponowano w [2]). Można ją przeprowadzić na wiele sposobów, na przykład definiując:

```
type
    vector_of_elements  = array[1..maxCol] of ElementType;
    vector_of_pointers  = array[1..maxRow] of pointer;
```

a następnie deklarując:

```
var
    A : pointer;
```

Element a_{ij} jest identyfikowany przez:
vector_of_elements (vector_of_pointers (A^) [i^]) j.

Dokonując następnie zmian w powyższych definicjach i deklaracji, a zatem definiując:

```
type
ptr_to_vector  = ^vector_of_elements;
ptr_to_matrix  = ^vector_of_pointers;
```

przy czym nadal

```
var
    A : pointer;
```

element a_{ij} jest identyfikowany przez

ptr_to_vector (ptr_to_matrix (A)^[i])^[j].

W celu uniknięcia odwzorowań typów, należy użyć definicji:

```
type
    vector_of_elements     = array[1..maxCol] of ElementType;
    ptr_to_vector          =^vector_of_elements;
    vector_of_pointers     = array[1..maxRow] of ptr_to_vector;
    ptr_to_matrix          =^vector_of_pointers;
```

oraz zadeklarować

```
var
    A : ptr_to_matrix;
```

Element a_{ij} jest teraz identyfikowany przez
A^ [i]^[j].

Wprowadzając powyższe definicje i deklarację zmiennej wskaźnikowej A, łatwo zmodyfikować wszystkie programy, zastępując zapis A[i][j] zapisem A^[i]^[j]. Oczywiście długość wiersza macierzy nie może przekroczyć 65 520 bajtów, co w przypadku

```
ElementType = extended;
```

oznacza maksymalnie 6552 elementy.

Następna implementacja macierzy, która jest uogólnieniem implementacji zawartej w [2], pozwala również ominąć to ograniczenie. Przedstawiona na wydruku 2 funkcja GRAM_SCHMIDT, która wykonuje ortonormalizację Grama-Schmidta jest przykładem zastosowania wyżej przedstawionej koncepcji.

```
function GRAM_SCHMIDT( A   : ptr_to_matrix;
                       m,
                       n   : dimension
                     ) : boolean;
var
    i, j, l : integer;
    s, p    : extended;

begin
  for i := 1 to m do begin
    s := 0.0;
    for l := 1 to i-1 do begin
      p := 0.0;
      for j := 1 to n do
        p := p + A^[i]^[j] * A^[l]^[j];
      for j := 1 to n do
        A^[i]^[j] := A^[i]^[j] - p * A^[l]^[j]
    end;
    for j := 1 to n do
      s := s + sqr(A^[i]^[j]);
    s := sqrt( s );
    if s <= 0.1e-9 then begin
      GRAM_SCHMIDT := false;
      Exit;
    end;
    for j := 1 to n do
      A^[i]^[j] := A^[i]^[j] / s
  end;
  GRAM_SCHMIDT := true;
end;
```

Wydruk 2. Funkcja wykonująca ortogonalizację Grama-Schmidta

W przypadku implementacji wielu macierzy o elementach różnego typu można wprowadzić definicje bardziej szczegółowe, na przykład:

```
vector_of_int_elements     =  array[1..maxCol] of integer;
ptr_to_int_vector          =  ^vector_of_int_elements;
vector_of_pointers_to_int  =  array[1..maxRow] of ptr_to_int_vector;
ptr_to_int_matrix          = ^vector_of_pointers_to_int;
```

IMPLEMENTACJA MACIERZY ZA POMOCĄ DŁUGIEGO WEKTORA

Przedstawiona na wydruku 3 implementacja macierzy w Turbo Pascalu 5.0 jest uogólnieniem implementacji wektora przedstawionej w [2]. Rozmiar pamięci przydzielonej obiektowi macierzy A_{mn} będzie zależeć tylko od liczby wierszy *m* i liczby kolumn *n*. Pozwoli to efektywniej gospodarować wolną pamięcią komputera w programach

```
type
    t_size  =  word;

const
    BlockSize : word  = $FFF0;
    SegAddr   : word  = $0FFF;
    MaxBlocks         = 25;

function ALLOC_MEMORY( n : longint; element_size : t_size
                     ) : pointer;

var
    p                 : pointer;
    vptr              : array[1..MaxBlocks] of pointer;
    nBlocks, i        : integer;
    contiguous_memory,
    memalloc          : boolean;
    size              : longint;
    lastBlockSize     : word;

begin
  size          := n * element_size ;
  nBlocks       := (size - 1) div BlockSize + 1;
  lastBlockSize := word( size - ( nBlocks-1 ) * BlockSize );
  if size > MemAvail then begin
    writeln( 'Insufficient memory!' );
    ALLOC_MEMORY := NIL;
    Exit;
  end;
  if nBlocks > maxBlocks then begin
    writeln( 'Too many blocks of memory required!' );
    ALLOC_MEMORY := NIL;
    Exit;
  end;
  for i := 1 to nBlocks do vptr[i] := NIL; memalloc := true;
  for i := 1 to nBlocks-1 do begin
    vptr[ i ] := MyGetMem( BlockSize );
    if vptr[ i ] = NIL then memalloc := false;
  end;
  vptr[ nBlocks ] := MyGetMem( lastBlockSize );
  if vptr[ nBlocks ] = NIL then memalloc := false;
  if not memalloc then begin
    for i := 1 to nBlocks-1 do
      if vptr[ i ] <> NIL then
        MyFreeMem( vptr[ i ], BlockSize);
    if vptr[ nBlocks ] <> NIL then
      MyFreeMem( vptr[ nBlocks ], lastBlockSize );
    ALLOC_MEMORY := NIL;
    Exit;
  end;
  contiguous_memory := true;
  for i := 1 to nBlocks - 1 do
    if PtrStruc( vptr[ i ] ).seg + SegAddr <>
       PtrStruc( vptr[ i + 1 ] ).seg then begin
      contiguous_memory := false;
      writeln( 'Not contiguous memory allocated!' );
      writeln( 'Block ', i, ' Seg ', PtrStruc( vptr[ i ] ).seg : 4,
               '   Ofs ', PtrStruc( vptr[ i ] ).ofs : 4              );
      writeln( 'Block ', i + 1, ' Seg ',
               PtrStruc( vptr[ i + 1 ] ).seg : 4, '   Ofs ',
               PtrStruc( vptr[ i + 1 ] ).ofs : 4                    );
  end;
  if contiguous_memory then begin
    ALLOC_MEMORY := vptr[ 1 ];
    Exit;
  end;
  for i := 1 to nBlocks - 1 do
    MyFreeMem( vptr[ i ], BlockSize );
  MyFreeMem( vptr[ nBlocks ], lastBlockSize );
  ALLOC_MEMORY := NIL;
end;

function GET_PTR_to_Mat_ELEMENT( base_ptr : pointer;
                                 i,j : i_jth_element;
                                 numCols : dimension;
                                 element_size : t_size )
                               : pointer;
var
    segs_, ofs_ : word;
    membytes    : longint;

begin
  membytes := longint( i * ( numCols + 1 ) + j ) *
                       element_size;
  segs_    := word( membytes div BlockSize       );
  ofs_     := word( membytes-segs_ * BlockSize   );
  Inc( PtrStruc( base_ptr ).seg, segs_ * SegAddr );
  Inc( PtrStruc( base_ptr ).ofs, ofs_            );
  GET_PTR_to_Mat_ELEMENT := base_ptr;
end;

function GET_PTR_to_ROW( base_ptr     : pointer;
                         i            : ith_element;
                         numCols      : dimension;
                         element_size : t_size
                       ) : pointer;
var
    segs_, ofs_ : word;
    membytes    : longint;

begin
  membytes := longint( i * ( numCols + 1 ) * element_size );
  segs_    := word( membytes div BlockSize       );
  ofs_     := word( membytes-segs_ * BlockSize   );
  Inc( PtrStruc( base_ptr ).seg, segs_ * SegAddr );
  Inc( PtrStruc( base_ptr ).ofs, ofs_            );
  GET_PTR_to_ROW := base_ptr;
end;


procedure FREE_MEMORY( p            : pointer;
                       n            : longint;
                       element_size : t_size     );

var
    nBlocks, i    : integer;
    lastBlockSize : word;
    size          : longint;
```

```
begin
  if p = NIL then Exit;
  size          := n * element_size;
  nBlocks       := ( size - 1 ) div BlockSize + 1;
  lastBlockSize := word( size - ( nBlocks - 1 ) * BlockSize);
  for i := 1 to nBlocks - 1 do begin
    MyFreeMem( p, BlockSize );
    inc( PtrStruc( p ).seg, SegAddr );
  end;
  MyFreeMem( p, lastBlockSize );
end;
```

Wydruk 3. Implementacja podstawowych operacji na dynamicznie alokowanej macierzy

wykorzystujących implementacje macierzy. Implementację macierzy można przeprowadzić korzystając z poniższych operacji.

ALLOC_MEMORY (n : longint; element_size : t_size
) : pointer;

Przydziela pamięć obiektowi o *n* elementach długości element_size, wskazywanemu przez zmienną wskaźnikową (jest ona rozszerzeniem operacji przedstawionej w [2]). Elementy są indeksowane od zera.

GET_PTR_to_Mat_ELEMENT (base_ptr : pointer;
i, j : i_jth_element;
numCols : dimension;
element_size : t_size
) : pointer;

Dostarcza wskaźnik na element *(i, j)* macierzy.

GET_PTR_to_ROW (base_ptr : pointer;
i : ith_element;
numCols : dimension;
element_size : t_size
) : pointer;

Dostarcza wskaźnik na *i*-ty wiersz macierzy.

FREE_MEMORY (p : pointer;
n : longint;
element_size : t_size);

Zwalnia pamięć przydzieloną obiektowi za pomocą funkcji ALLOC_MEMORY.

Operacje GET wykorzystują jako parametr *n* – liczbę kolumn macierzy (parametr formalny numCols). Macierz jest zapisana w pamięci jako długi wektor – wierszami, co jest zgodne z implementacją macierzy jako tablicy dwuwymiarowej. Na wydruku 4 przedstawiono implementację operacji odwracania macierzy, wykorzystując powyższe definicje.

```
function INVERSE(   base_ptr_A : pointer;
                    n          : dimension
                ) : ElementType;

label
    Exit_;

var
    cols,
    rows            : ptr_to_int_vector;
    Q               : ptr_to_bool_vector;
    P               : ptr_to_ext_vector;
    col, row        : word;
    max, temp, det  : ElementType;
    i, j, k, l      : integer;

begin
  det := 1.0;
  cols := ALLOC_MEMORY( n+1,  int_size  );
  rows := ALLOC_MEMORY( n+1,  int_size  );
  Q    := ALLOC_MEMORY( n+1,  bool_size );
  P    := ALLOC_MEMORY( n+1,  ext_size  );
  if ( cols = NIL ) or ( rows = NIL ) or
     (    Q = NIL ) or ( P    = NIL     ) then begin
    det := 0.0;
    writeln( 'Insufficient memory!' );
    goto Exit_;
  end;
  for i :=1 to n do
    Q^[ i ] := false;
  for i := 1 to n do begin
    max := 0.0; col := 0; row := 0;
    for j := 1 to n do if not  Q^[ j ]  then begin
      for k := 1 to n do if not  Q^[ k ]  and
        ( abs( max ) <= abs(
          ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, j, k, n, elem_size )^ ) ) )
        then begin
        col := k; row := j;
        max := ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, j, k, n, elem_size )^ ) ;
      end
    end;
    if ( abs( max ) <= 1.0e-9 ) or ( col * row = 0 ) then begin
      det := 0.0;
      goto Exit_;
    end;
  Q^[ col ] := true;
  if col <> row then begin
    det := - det;
    for j := 1 to n do begin
      temp :=
        ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, row, j, n, elem_size )^ ) ;
      ElementType( GET_PTR_to_Mat_ELEMENT(
                     base_ptr_A, row, j, n, elem_size )^ ) :=
            ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, j, n, elem_size )^ );
       ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, j, n, elem_size )^ ) :=
         temp;
    end
  end;
  rows^[ i ] := row;
  cols^[ i ] := col;
  P^[ i ]
    := ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, col, n, elem_size )^ );
  det := det *
         ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, col, n, elem_size )^ );
  ElementType( GET_PTR_to_Mat_ELEMENT(
                  base_ptr_A, col, col, n, elem_size )^ ) := 1.0;
  for j := 1 to n do
    ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, j, n, elem_size )^ ) :=
        ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, col, j, n, elem_size )^ ) /
        P^[ i ];
  for l := 1 to n do  if l <> col then begin
    temp :=
      ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, l, col, n, elem_size )^ );
    ElementType( GET_PTR_to_Mat_ELEMENT(
                  base_ptr_A, l, col, n, elem_size )^ ) := 0.0;
    for j := 1 to n do
      ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, l, j, n, elem_size )^ ) :=
          ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, l, j, n, elem_size )^ ) -
          ElementType( GET_PTR_to_Mat_ELEMENT(
                  base_ptr_A, col, j, n, elem_size )^ ) * temp;
  end
end;
for j := n downto 1 do if rows^[ j ] <> cols^[ j ] then begin
  col := cols^[ j ];
  row := rows^[ j ];
  for l :=1 to n do begin
    temp :=
      ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, l, row, n, elem_size )^ );

    ElementType( GET_PTR_to_Mat_ELEMENT(
                  base_ptr_A, l, row, n, elem_size )^ ) :=
          ElementType( GET_PTR_to_Mat_ELEMENT(
                       base_ptr_A, l, col, n, elem_size )^ );
    ElementType( GET_PTR_to_Mat_ELEMENT(
                  base_ptr_A, l, col, n, elem_size )^ ) :=
        temp;
  end
end;
Exit_:
FREE_MEMORY( rows, n + 1, int_size  );
FREE_MEMORY( cols, n + 1, int_size  );
FREE_MEMORY( Q,    n + 1, bool_size );
FREE_MEMORY( P,    n + 1, ext_size  );
INVERSE := det;
end;
```

Wydruk 4. Implementacja operacji odwracania macierzy

Użyte w tekście definicje oraz implementacje operacji są zawarte w jednostce LargeMatrix, opracowanej przez autora w celu rozszerzenia możliwości implementacji algorytmów w języku Turbo Pascal 5.0. Jednostka LargeMatrix zawiera również rozszerzoną wersję jednostki LongRArr, przedstawionej w [2]. Cała jednostka LargeMatrix zajmuje obecnie 1102 wiersze, z tych więc względów nie jest w tym artykule przedstawiona w całości. Alokacja pamięci z wolnej pamięci komputera tzw. sterty (ang. *heap memory*) oraz operowanie wskaźnikami wydłuża czas pracy programów, ale zwiększa jednak dość istotnie możliwości implementacji algorytmów. Turbo Pascal 5.0 nie ustępuje więc językowi C. Również w języku C programy kompilowane z modelem pamięci *large* i *huge* charakteryzują się wydłużonym czasem wykonania ze względu na operacje na wskaźnikach. W Turbo Pascalu 5.0 możliwe jest wykonywanie operacji na wskaźnikach, chociaż nie w taki prosty sposób jak w języku C. Przedstawiona na wydruku 5 jednostka LargeMatrix zawiera odpowiednie implementacje operacji zwiększania i zmniejszania wskaźnika.

Korzystają z funkcji PTR_INCREMENT, iloczyn wektorów o elementach typu *real* można wyznaczyć jak w implementacji tej operacji przedstawionej na wydruku 6.

```
const
      BlockSize : word   = $FFF0;
      SegAddr   : word   = $0FFF;
      MaxBlocks          = 25;

type
     PtrStruc = record
       ofs, seg : word;
     end;
function PTR_INCREMENT( p : pointer; size : word ) : pointer;

 var
    new_ofs : longint;

 begin
   if longint( PtrStruc( p ).ofs + size ) >= BlockSize then begin
     new_ofs := longint( PtrStruc( p ).ofs ) +
                longint( size ) - BlockSize;
     Inc(PtrStruc( p ).seg, SegAddr );
     PtrStruc(p).ofs := word( new_ofs );
   end
   else Inc( PtrStruc(p).ofs, size);
   Ptr_increment := p;
 end;

function PTR_DECREMENT( p : pointer; size : word ) : pointer;

 var
    new_ofs : longint;

 begin
   if longint( PtrStruc( p ).ofs - size ) < 0 then begin
     new_ofs := longint( PtrStruc( p ).ofs ) -
                size + BlockSize;
     Dec( PtrStruc( p ).seg, SegAddr );
     PtrStruc( p ).ofs := word( new_ofs );
   end
   else Dec( PtrStruc( p ).ofs, size);
   PTR_DECREMENT := p;
 end;
```

Wydruk 5. Implementacja operacji zwiększania i zmniejszania wskaźnika w Turbo Pascalu 5.0

Mimo tego, że Turbo Pascal 5.0 nie jest tak ukierunkowany na operowanie wolną pamięcią komputera jak język C, to przy umiejętnym wykorzystywaniu możliwości jest on nadal atrakcyjnym narzędziem implementacji algorytmów. Natomiast pod względem odwzorowań typów znacznie przewyższa, jak dotąd, możliwości języka C na mikrokomputerach.

```
 const
      real_size : word = SizeOf( real );

function real_SCALAR_PRODUCT(  base_ptr_x,
                               base_ptr_y   : pointer;
                               n            : dimension
                            ) : real;
 var
     j : longint;
     s : extended;

 begin
   s := 0.0;
   for j := 1 to n do begin
     base_ptr_x := PTR_INCREMENT( base_ptr_x, real_size );

   base_ptr_y := PTR_INCREMENT( base_ptr_y, real_size );
   s := s + ptr_to_real( base_ptr_x )^   *
            ptr_to_real( base_ptr_y )^   ;
 end;
 real_SCALAR_PRODUCT := s;
end;
```

Wydruk 6. Implementacja operacji – iloczyn wektorów

LITERATURA

[1] Aho A. V., Hopcroft J. E., Ullman J. E.: Data Structures and Algorithms. Addison-Wesley, Reading (MA), 1983

[2] Runka H. J.: Długi wektor w Turbo Pascalu 5.0. Informatyka nr 2, 1990

[3] Runka H. J.: Wstęp do teorii algorytmów. Struktury danych w projektowaniu algorytmów na grafach. Akademia Ekonomiczna, Poznań, 1989

[4] Turbo Pascal Reference Guide, Version 5.0. Borland International, 1988.

Przypominamy, że zlecenia na umieszczanie ogłoszeń w INFORMATYCE przyjmuje w bieżącym roku bezpośrednio nasza redakcja. Czekamy na zgłoszenia.

Symboliczny program uruchomieniowy dla Ady (2)

W drugiej części artykułu omówiono mechanizm kontekstu, akcje oraz inne właściwości programu.

KONTEKST

Mechanizm kontekstu służy do określenia części programu podlegającej testowaniu. Może on również określać każde środowisko, w którym należy wyspecyfikować zbiór obiektów tworzonych przez fragment programu. Kontekst określa ciało (każdą jednostkę Ady, która może zawierać własne procedury obsługi sytuacji wyjątkowych) i jest identyfikowany przez nazwę.

Definicja dowolnego punktu widoczności z dowolnego kontekstu programu wymaga wprowadzenia specyficznych notacji, pozwalających na dostęp do nazwy Ady spoza zasięgu normalnej jej widoczności.

Kontekst może być również określony przez każdą instrukcję bezpośrednio należącą do niego, co sprowadza się do określania kontekstów przez proste „wskazywanie" ich.

Arytmetyka kontekstów

Arytmetykę kontekstów ograniczono do niezbędnego minimum, jest ona jednak konieczna do odróżniania ciała programu od obiektu typu zadaniowego, ponieważ obiekty, na których działają zadania, mogą być różne w każdej chwili. Także relacje między obiektami zdefiniowane w specyfikacji zdarzenia dotyczą jedynie obiektów chwilowych, a nie bezwzględnych. Ten sam problem występuje w przypadku procedur rekurencyjnych. Kontekst może być więc określony pośrednio przy użyciu dwóch operatorów działających na innych kontekstach:
〉 definiuje kontekst na podstawie kontekstu uaktywnienia (master);
〈 definiuje kontekst na podstawie kontekstu przezeń wywołanego.

Operatory te nie wystarczają do zdefiniowania pełnego zbioru kontekstów dynamicznych, umożliwiają jednak specyfikację każdego kontekstu tylko przy użyciu relacji między nimi, zgodnie z językiem Ada.

Przykłady

P.T 4.0	**T** jest ładowana w **P**
P 〉 TT	zadania typu **TT** uaktywniane przez **P**
P 〈 Q	procedura **Q** jest widzialna z procedury wywołującej **P**

AKCJE

Wyświetlanie

Głównym celem wyświetlania jest umożliwienie użytkownikowi dokonania szybkiej diagnozy wówczas, gdy sama specyfikacja zdarzenia nie zawiera wystarczających informacji.
Polecenie wyświetlania może mieć następujące postaci:

Display [all] [invalid : undef] [with access] [display list] [of context] – lista wartości obiektów lub ich atrybutów;

Display [all] [invalid : undef] [with access] [type list] [of context] – lista wartości obiektów wybranego typu;

Display [all] [invalid : undef] [with access] [of context] – lista wartości obiektów;

Display [all] task [with access] [task_list] [of context] – dokładny stan zadań bieżących lub wybranego typu;

Display [all] task [with access] [of context] – dokładny stan zbioru zadań.

Jednym z prostszych przypadków jest wyświetlanie wartości prostych, np. przy użyciu polecenia **Display I** lub **Display T'First** stosowane głównie przy badaniu już zlokalizowanych błędów. Opcja **invalid** lub **undef** służy do wybrania tych spośród ustalonego zbioru obiektów, które nie spełniają ograniczeń nałożonych przez ich podtypy lub są niereprezentowalne. Umożliwia to wychwycenie obiektów nie zainicjowanych lub takich, którym przypisano wartości nie zainicjowane.

Opcja **with access** wskazuje, że należy wyświetlić obiekty wskazywane przez aktualne wartości zmiennych wskazujących. Pozwala to na wyświetlanie całych struktur lub kompletnej struktury zadań.

Opcja **all** wskazuje, że wyświetleniu podlegają wszystkie obiekty dostępne, a nie tylko obiekty widoczne w danym kontekście.

Opcja **of context** pozwala na wyświetlenie tylko obiektów należących wprost do wybranego kontekstu.

Wybór przez typ jest oparty na fakcie narzucenia w języku Ada sztywnych reguł na operację przypisania; taki wybór pozwala ograniczyć zbiór wyświetlanych obiektów do tych, którym przypisano wartości danego typu.

Wyświetlanie obiektów jest zawsze symboliczne (z wyjątkiem obiektów **undefined**), niezależnie od ich typu, przy użyciu konwencji reprezentacji Ady, z uwzględnieniem właściwości i atrybutów tych wartości, niezależnie od jawnej klauzuli reprezentacji.

Wyświetlanie zadań umożliwia wykrycie błędów synchronizacji między zadaniami, zwłaszcza zakleszczeń. Stan zawieszonych zadań jest odtwarzany i wyświetlany z uwzględnieniem zależności między poszczególnymi zadaniami. Wyświetlany stan nie jest związany z fizyczną realizacją zadania.

Wizualizacja sytuacji zablokowania, częściowego lub całkowitego, jest realizowana przez polecenie typu: **display all task with access.** Nierealizowalne frazy **accept** są zaznaczone podczas wyświetlania po wykryciu odpowiadających im cykli. Właściwy test może być wywołany przez wykrycie sytuacji anormalnej.

Przykłady

Display all invalid positive – wyświetlenie wszystkich zmiennych zadeklarowanych, jako dodatnie o wartościach mniejszych lub równych 0;

Display task truc – wyświetlanie wszystkich widocznych zadań typu **truc**;

Display all I of P – wyświetlenie wartości **I** dla wszystkich aktywnych wcieleń procedury **P**;

Display with access my_list – wyświetlenie wartości wszystkich elementów listy **my_list**;

Display task with access T (J) – pokazanie przyczyny zablokowania **T (J)**;

Display – wyświetlenie wszystkich obiektów widocznych.

Wyświetlanie zdarzeń

Wyświetlanie zdarzeń umożliwia pośrednie opisanie dynamicznej zawartości określonego zdarzenia, zależnie od sposobu jego specyfikacji.

Jeśli test jest globalny, to nie zawsze zdarzenie zachodzi w kontekście, w którym klauzula jest specyfikowana. Znana jest droga łącząca punkt

zajścia zdarzenia z kontekstem zawierającym klauzulę. Dla ustalenia uwagi, jeżeli kontekstem tym jest zbiór programu i jeśli istnieje tylko jedno zadanie, to droga ta jest równoważna śladowi zawartości stosu.

W przypadku zdarzeń sekwencyjnych (zawierających **times** lub **then**) znane są ślady poszczególnych zdarzeń. Wygląda to tak, jak śledzenie sytuacji błędnych, w których nie jest znana dokładna reakcja na daną sytuację. Jeśli zdarzenie stanowi klasę instrukcji, to parametry instrukcji mogą być istotne (np. procedura wywoływana przez **call**); w takim przypadku można wykorzystać informacje uprzednio niejawnie zapamiętane. Dla obiektów dynamicznych zostaje zapamiętany ślad ich wartości i kolejne wartości mogą być wyświetlane. Dla zdarzeń (np. sytuacji wyjątkowych) – są wyświetlane ich nazwy. Liczba wyświetlanych komunikatów odpowiada liczbie możliwych konstrukcji klauzul, co ułatwia użytkownikowi posługiwanie się programem.

Interakcja z użytkownikiem

Interakcja programu z użytkownikiem jest realizowana przez instrukcję **accept**. Może ona stanowić część klauzuli testu – jest to domyślne zachowanie w środowisku konwersacyjnym. Wykonanie takiej akcji polega na przyjęciu grupy poleceń do wykonania. Przebieg tej komunikacji jest uzależniony od kolejno wprowadzanych poleceń.

Komunikacja z otoczeniem

1. Modyfikacja środowiska zewnętrznego

Komunikacja bezpośrednia polega w tym przypadku na przesyłaniu informacji poza mechanizmami systemu operacyjnego. Dokładny zakres możliwości takiego mechanizmu zależy od konkretnego systemu. Szczególnie duże możliwości udostępnia tutaj system Unix.
Komunikacja pośrednia natomiast polega na tym, że dla każdej akcji jest możliwe określenie logicznego urządzenia wejścia–wyjścia. Są tu wykorzystywane pełne możliwości współczesnych systemów obsługi urządzeń, w szczególności środowisko konwersacyjne.

2. Interakcja z narzędziami przetwarzania plików źródłowych
Komunikacja bezpośrednia polega na przekazaniu polecenia nie interpretowanego przez system do części systemu zajmującej się przetwarzaniem tekstów (np. edytor tekstu). Początkowo nazwa pliku przetwarzanego i położenie tekstu w pliku nie są określone. Parametry te są przekazywane do edytora w chwili wykrycia błędu przez program uruchomieniowy.
Komunikacja pośrednia: działaniu programu uruchomieniowego towarzyszy wyświetlanie tekstu źródłowego, co umożliwia lepszy wgląd w strukturę programu i korzystanie podczas uruchamiania z komentarzy umieszczonych w programie. Dodatkowo można określić każde miejsce programu poprzez wskazanie go w pliku źródłowym.

Wybrane zastosowania

Mechanizmy opisane poniżej mają niewielkie znaczenie w procesie uruchamiania programu, są one wykorzystywane głównie w końcowym testowaniu programu i umożliwiają obserwowanie jego działania.

- Śledzenie

Polecenie **keep** zawiera wszystkie opcje polecenia **display**, jednak wyniki jego działania są przeznaczone do analizy programowej. Niektóre postaci tego polecenia powodują wysyłanie zebranych danych do pamięci masowej, co umożliwia monitorowanie programu w czasie rzeczywistym.

- Pomiary czasu

Podstawowa postać polecenia **time in context** umożliwia pomiar czasu zużytego przez program w danym kontekście, zwykle z uwzględnieniem danych statystycznych, np. liczby uaktywnień średniego czasu jednorazowo zużytego w kontekście itp. Wyświetlanie może wiązać się z zajściem konkretnego zdarzenia; możliwe są równoczesne pomiary w wielu kontekstach. Zakres możliwości specyfikacji kontekstu daje wyobrażenie o mocy tego mechanizmu.

Dwie powyższe funkcje są niezbędne do zebrania danych o programie. Poniżej podano kilka przykładów ilustrujących możliwości ich użycia.

time in SP	prosty pomiar wydajności
when delay in T1 time in T2, T3, T4	złożony pomiar wydajności
when call or return = > keep	statystyka wywołań i określenie poziomu zagłębienia
when goto or call or raise = > keep	testy niesekwencyjności
when call or accept = > keep	
T'Storage-size	użycie stosów
when call (SP) = > keep	
calendar time	zapamiętanie czasu akcji

Komunikacja między narzędziami

Komunikacja ta jest realizowana jako kombinacja uprzednio prezentowanych możliwości. W rzeczywistości jest możliwa tylko komunikacja z narzędziem obróbki tekstów. Komunikację ze wszystkimi innymi składnikami uzyskuje się za jej pośrednictwem, przez dynamiczne zarządzanie strumieniami wejścia–wyjścia, bezpośrednią komunikację ze środowiskiem oraz wyprowadzanie informacji poleceniem **keep**. Dzięki takiej realizacji komunikacji, nie zależy ona bezpośrednio od użytkownika, co zmniejsza ilość przesyłanych informacji oraz ryzyko błędów podczas ich przekazywania i obróbki.

Inne właściwości systemu

- Skróty

Jest możliwe wprowadzanie i stosowanie skrótów nazw obiektów lub wyrażeń. Przyspiesza to wprowadzanie poleceń przez użytkownika i pozwala efektywnie wykorzystać klawisze specjalne (funkcyjne) na klawiaturze terminala.

- Zarządzanie testami

Oprócz możliwości dynamicznego specyfikowania testów, użytkownik może wykończyć wykonywanie dowolnego testu lub grupy testów przez określenie w poleceniu zakończenia testów obiektów, których dotyczyły te testy. Dynamiczne modyfikowanie testu może odbywać się przez określenie obiektu występującego jedynie w zbiorze objętym testem przez polecenie pominięcia testu w określonym kontekście.

- Wyświetlanie aktualnego zbioru testów

Z punktu widzenia użytkownika efekt wyświetlania testów przypomina efekt ich zakończenia. Jest możliwe dynamiczne wyświetlanie tekstu, którego dotyczy test, oraz aktualnego wyniku testu, dzięki czemu użytkownik w każdej chwili może zapoznać się ze stanem operacji śledzenia programu.

Przykłady

Skróty:

```
? = display
dump = keep all with access
Ac = in this_task (i)                    – wskazane zadanie;
EcT = when accept (e3 (j)) in Ac         – wskazane entry;
see = (display; ed; accept)
```

Zakończenie testów:

```
terminate Ac                – usunięcie Ac i EcT;
terminate P of T            – usunięcie testów dotyczących P
                              tylko w zadaniu T.
```

Wyświetlanie testów:

```
call_trace = when 100 times (call or return)
when others => use of call_trace
when call_trace => null
```

Efekty optymalizacji kodu wynikowego

Realizacja przedstawionego programu odbiega w wielu punktach od

typowych realizacji programów uruchomieniowych działających na poziomie asemblera i nie wymaga współdziałania z innymi programami systemu w procesie uruchamiania programu. Wynika to z następujących założeń:
– analizowane programy są tworzone przez kompilator;
– może istnieć kilka alternatywnych implementacji egzekutora,
– opcjonalnie można wykorzystać charakterystyczne właściwości sprzętu;
– dozwolona jest optymalizacja struktury kompilatora i struktury generowanego kodu.

Ostatecznie określono zbiór podstawowych mechanizmów, który nakłada minimalne ograniczenia na postać przetwarzanego kodu, ale uwzględnia zgodność postaci programu z regułami zdefiniowanymi przez standard języka.

Opisane funkcje działają niezależnie od stopnia optymalizacji kodu programu dokonywanej przez kompilator. Występujące ograniczenia są związane głównie ze strukturą danego programu lub właściwościami konkretnego programu uruchomieniowego.
Są one następujące:

- każda akcja uznana przez kompilator za nadmiarową (i ewentualnie usunięta) nie może być testowana ani użyta pośrednio przez program uruchomieniowy;
- żadne założeniowe kompilatora dotyczące programu nie może zostać naruszone przez program uruchomieniowy,
- informacje niezbędne do przekształcenia reprezentacji źródłowej programu w reprezentację kodu wynikowego są przechowywane tylko w takiej ilości, aby ich objętość była liniowo zależna od rozmiaru programu.

Możliwe jest określenie zależności zaproponowanych funkcji od powyższych założeń. W przypadku naruszenia któregoś z tych ograniczeń, użytkownik otrzyma stosowny komunikat.

* * *

W przypadku języka takiego jak Ada, w którym jawnie lub pośrednio może istnieć wiele reprezentacji użytych obiektów, dekodowanie reprezentacji maszynowych wymaga znajomości sposobu wewnętrznego działania kompilatora i narzędzia realizacji. Z drugiej strony, tłumaczenie pewnych zwrotów języka w implementacjach na niektóre komputery może być bardzo utrudnione i wymagać szczególnej postaci kodu wynikowego. Dekodowanie zachowania sekwencyjnego maszyny również wymaga wielu informacji o sposobie działania kompilatora i narzędzia realizacji.

Wydaje się więc użyteczne dostarczenie narzędzia dostosowanego do takiej sytuacji testującego niezmiennik programu Ada: jego zachowanie logiczne.

Tłumaczyła
DOROTA INKIELMAN

Nowe książki

WYDAWNICTWA NAUKOWO-TECHNICZNE

Lech Banachowski, Antoni Kreczmar, Wojciech Rytter: Analiza algorytmów i struktur danych. Warszawa 1989, nakład 5000 egz., s. 220, ISBN 83-204-1148-3

W książce rozważa się strukturę i własności algorytmu z różnych punktów widzenia i na różnych poziomach abstrakcji. Autorzy koncentrują się na metodach analizy algorytmów i aspekcie probabilistycznym, strukturalnym i symulacyjnym. Pokazują użyteczność metod matematycznych w rozumieniu struktury algorytmu i problemu algorytmicznego.
Książka jest przeznaczona dla programistów, projektantów systemów przetwarzania informacji, pracowników nauki zajmujących się informatyką oraz dla studentów kierunków informatycznych i matematycznych.

Bohdan Frelek: Commodore 64. Warszawa 1988, nakład 25 tys. egz., s. 208, ISBN 83-204-0990-X

W książce podano podstawowe informacje dotyczące mikrokomputera Commodore (C64). Omówiono jego budowę, wykorzystanie oraz współpracę z urządzeniami zewnętrznymi. Opisano sposób programowania w Basicu i asemblerze. Przedstawiono możliwości uzyskania efektów dźwiękowych i graficznych. Podano również wiadomości o komputerach pokrewnych, takich jak: VC 20, C 16, Plus/4, C 128.
Książka jest przeznaczona dla szerokiego kręgu czytelników interesujących się użytkowaniem i programowaniem popularnych mikrokomputerów osobistych.

Robert Laurence Baber: O programowaniu inaczej. Warszawa 1989, nakład 6000 egz., s. 172, ISBN 83-204-1072-X

Jest to książka o programowaniu – o tym, jak ono powstaje, jak są od niego uzależnione wszelkie zastosowania komputerów, jakie grzechy ciążą na istniejącym oprogramowaniu i skąd się wywodzą. O tym wreszcie, jakimi drogami może się potoczyć dalszy jego rozwój. Zamieszczony na końcu obszerny dodatek, ułożony w formie odpowiedzi na pytania testu jest małą encyklopedią oprogramowania. Autor, mimo że porusza zagadnienia dużej wagi, prowadzi narrację w formie gawędy przeplatanej ciekawostkami anegdotami.
Książka jest przeznaczona dla szerokiego kręgu odbiorców – dla wszystkich zainteresowanych informatyką.

ASSOCIATION FOR COMPUTING MACHINERY
1991 COMPUTER SCIENCE CONFERENCE

Konferencja ta przygotowana przez ACM odbędzie się w dniach 5–7 marca 1991 r. w San Antonio w Texasie. Jej treścią będą przewidywane potrzeby zbliżającego się 21. wieku w zakresie nowych technologii i badań naukowych w dziedzinie informatyki stosowanej. Program obejmuje trzy podstawowe zagadnienia: przyszłe technologie, rezultaty badań, systemy prototypowe oraz nowe rozwiązania.

Tematyka została, podzielona na część referatową, afiszową oraz wykładową i obejmuje następujące obszary tematyczne:
☐ Systemy: baz danych, transakcyjne, rozproszone, plików odpornych na błędy, czasu rzeczywistego oraz komunikacyjne.
☐ Architektura: urządzeń optycznych, maszyn równoległych, logiki redukcyjnej oraz innych nowoczesnych maszyn.
☐ Teoria obliczeń: złożoności obliczeniowej, modeli obliczeń równoległych i rozproszonych oraz programowania w logice.
☐ Oprogramowanie: techniki i praktyki inżynierii oprogramowania, języków i kompilatorów.
☐ Współpraca z użytkownikiem: systemów okienkowych, programowania automatycznego, technik wizualizacji, technologii interakcji oraz przetwarzania zintegrowanego (Groupware).
☐ Aplikacja: w technice i nauce, handlu, wojsku, itp.
☐ Sztuczna inteligencja: robotyka, systemy uczące się, bazy wiedzy i systemy doradcze.

Referty nie powinny przekraczać 20 stron maszynopisu z podwójnym odstępem (ok. 5000 słów). Należy je wysłać w pięciu egzemplarzach przed 1 sierpnia 1990 roku. Autorzy zostaną zawiadomieni o przyjęciu referatu przed 1 listopada 1990 roku, a jego gotowy do powielenia tekst należy dostarczyć organizatorom przed 1 grudnia 1990 roku. Nie przyjęte referaty mogą być wykorzystane w CACM lub innych publikacjach wydawanych przez ACM.

Referaty powinny być przesłane pod adresem:

Richard Brice, MCC Corporation
3500 West Balcones, Center Drive, Austin, Texas 78759
(512) 338-3429

Bezpośrednio po tej konferencji, w dniach 7–8 marca 1991 r. odbędzie się konferencja ACM SIGCSE dotycząca problemów edukacji w informatyce, pod hasłem:

„Education, Research, Industry:
Keep the Information Flowing"

Bliższych informacji dotyczących tych konferencji udziela
dr inż. Wacław Iszkowski
Instytut Informatyki, Politechnika Warszawska,
ul. Nowowiejska 15/19,
00–655 Warszawa, tel. 21007-650

Japońskie projekty neurokomputerów

Rząd japoński i przedsiębiorstwa komputerowe ogłosiły wieloletni projekt opracowania neurokomputerów, zdolnych do złożonego, równoległego przetwarzania. Ministerstwo Handlu Zagranicznego i Przemysłu podało szczegóły finansowego wsparcia tego przedsięwzięcia o wartości 30 miliardów yenów. Projekt miał rozpocząć się w kwietniu 1989 roku i obejmować okres 9–10 lat. Uczestnictwo w nim zgłosiło sześć wielkich firm komputerowych: Fujitsu, Hitachi, Toshiba, Mitsubishi Electric, NEC i Oki Electric.

Komputer taki zawierałby około miliona elementów będących odpowiednikiem neuronów i byłby zdolny do dokładnego rozpoznawania znaków, głosów ludzkich, przestrzeni trójwymiarowej i innych elementów swego środowiska. Miałby też właściwości samonastawności i sterowania.

Zarówno NEC, jak i Fujitsu opracowują własne projekty neurokomputerów i współzawodniczą w tej dziedzinie. NEC zbudował już „neurokomputer osobisty" z możliwościami systemowymi odpowiadającymi superminikomputerowi. Zawiera on 82 tysiące neuronów i działa z szybkością 216 tysięcy połączeń na sekundę, wykorzystując metodę przetwarzania równoległego. Miał on pojawić się na rynku w cenie około 11 tysięcy dolarów i byłby to pierwszy sprzedawany neurokomputer. Oczekuje się, że będzie on kluczowym elementem w systemach eksportowych i systemów sterowania robotów.

Inny system tej firmy, który miał pojawić się na rynku w kwietniu 1989 roku, może rozpoznawać (z dokładnością 99,8%) 62 znaki alfanumeryczne, drukowane za pomocą mozaikowych drukarek uderzeniowych. Przy odczytywaniu 76 znaków i symboli w 12 krojach czcionek osiągnięto dokładność 99,95%. System rozpoznawania znaków jest oparty na metodzie powtarzającego się uczenia znaków, tzw. uczącym się algorytmie wstecznej propagacji. Dawne metody rozpoznawania znaków były oparte na metodach dopasowania wzoru lub analizy strukturalnej.

Działania Fujitsu w tej dziedzinie, wciąż w początkowych stadiach rozwoju, obejmują gry w rodzaju „złodzieje i policjanci", grane przez neurokomputerowe roboty. Firma opracowuje też kostkę neuronową, która ma posłużyć do budowy neurokomputerów.

Instytut Badawczy Mitsubishi rozpoczął ostatnio badania nad neurokomputerami oraz ich socjologicznymi i ekonomicznymi skutkami. W badaniach tych uczestniczy około 20 firm z przemysłu stalowego, samochodowego i innych.

Mikroprocesor 80486

W kwietniu 1989 r. firma Intel Corp. z Santa Clara w Kalifornii ogłosiła parametry swego nowego, 32-bitowego mikroprocesora 80486, którego wydajność obliczeniowa jest prawie czterokrotnie wyższa od mikroprocesora 80386. Wyposażony jest on w koprocesor sieci lokalnych LAN, a technika realizacji zapożyczona jest od układów ze zredukowaną listą rozkazów RISC. Nadaje się przede wszystkim do usprawnionego przetwarzania wieloprocesowego. Jest zgodny programowo z 80386 i istniejącym oprogramowaniem MS-DOS, a także z systemami UNIX V/386 i XENIX firmy Microsoft oraz OS/2 firmy IBM.

W wersji przedstawionej na targach Comdex '89 oferowano mikroprocesor z zegarem o częstotliwości 33 MHz („Byte" z grudnia 1989 r. podał informację Johna Crawforda, „głównego architekta" tego procesora, że jeszcze w 1989 r. pojawi się wersja 50 MHz; to samo źródło przewiduje realizację w ciągu 10 lat procesora zawierającego 50 mln tranzystorów).

W przeciwieństwie do układu 80386 układ 80486 obejmuje zespół zmiennego przecinka, zapewnia 8 Kbajtów szybkiej pamięci pomocniczej *(cache)* i pozwala na zarządzanie pamięcią, łącznie z dzieleniem na strony *(paging)*. Wraz z procesorem oferowane są też zestawy kostek w architekturze EISA *(Extended Industry Standard Architecture)* i mikrokanałowej *(Micro Channel)*, przyjętej m.in. w komputerach PS/2 firmy IBM. Próby z procesorem o zegarze 25 MHz potwierdziły osiągalną prędkość 15–20 MIPS (milionów operacji na sekundę). Koprocesor sieciowy pozwala na zastosowanie mikroprocesora do obsługi zbiorów.

W październiku 1989 r. podano, że w firmie Compaq Computer podczas rutynowego testowania pierwszych dostarczonych egzemplarzy mikroprocesora 80464 wykryto błąd w zespole zmiennego przecinka. Przewidywano, że spowoduje to co najmniej miesięczne opóźnienie masowych dostaw i konieczność wymiany dostarczonych już egzemplarzy. Compaq realizuje na tym procesorze komputer Deskpro 486/25, przeznaczony głównie do projektowania wspomaganego CAD. Również Hewlett-Packard podał dane swego systemu VECTRA 486 wykorzystującego ten procesor i architekturę EISA. Działa on o 1/3 szybciej od wersji wykorzystującej mikroprocesor 80386 o częstotliwości zegara 33 MHz.

Informacje napływające z Tajwanu mówią również o wielu opracowanych tam nowych modelach komputerów z zastosowaniem mikroprocesora 80486. Cena tego mikroprocesora przy większych partiach dostaw ustaliła się na poziomie ok. 950 dolarów.

DEC a rozwój sieci komputerowych

Ogłoszona ostatnio strategia zarządzania rozszerzonymi sieciami potwierdza, iż DEC przystępuje do prac nad sieciami komputerowymi obejmującymi całe przedsiębiorstwo, opartymi na międzynarodowych protokołach OSI (ang. *open system interconnection*). Ogłoszenie to pojawiło się ponad rok po ujawnieniu zamierzeń sieciowych IBM i innych firm. Stawia to firmę DEC w trudnej sytuacji, gdyż aby zdobyć klientów, musi zaoferować dodatkowe wyroby i usługi, często opracowane wspólnie z innymi firmami.

Proponowane przez DEC rozwiązanie nosi nazwę *Enterprise Management Architecture* (EMA – Architektura zarządzania przedsiębiorstwem) i opiera się na standardach międzynarodowych oraz wyrobach innych wytwórców. W dziedzinie sieci dalekosiężnych EMA powołuje się na standard X.25, ISDN (*Integrated Services Digital Network*), X.400, FTAM (*File Transfer Access and Management*) i inne protokoły OSI. Wykorzystano w niej też szybkie usługi transmisji cyfrowej równoważne T1 lub E1/G.703.

Aby zmniejszyć ręczne interwencje, zarządzanie siecią w standardzie EMA jest sterowane i monitorowane przez programy użytkowe, pośredniczące między systemami a jednostkami sieci. DEC zachęca też innych wytwórców do opracowywania urządzeń integrujących funkcje zarządzania. Dotychczas siedem firm zgłosiło swój udział – pięć z USA: Codex Corp. (oddział Motoroli), DCA Inc., Stratacom Inc., Timeplex Inc. i Vitalink Communications Corp., a także Siemens AG z RFN i TSB International Inc. z Kanady. Każda z nich opracowuje moduły dostępu zgodne z podręcznikiem EMA, który miał być opublikowany na początku 1989 roku. Pozwolą one na sterowanie i obserwację wybranych wyrobów tych firm.

DEC ma już tradycję w takich wspólnych przedsięwzięciach. Na razie nie jest przewidziana współpraca z SNA (ang. *System Network Architecture*) i IBM, co mogłoby rozszerzyć zasięg EMA. DEC negocjuje współpracę z innymi sprzedawcami.

Przyjęcie przez DEC standardów OSI może wywrzeć nacisk na IBM, by pełniej przyjmowała standardy międzynarodowe, do czego zmuszają ją zresztą wymagania rządowe. Również ostatnie dobre wyniki handlowe IBM w Europie potwierdzają ten kierunek rozwoju.

Drukarka laserowa HP Laser Jet IID

Jest to pierwsza drukarka o dość niskiej cenie (4295 dolarów), zapewniająca automatyczny wydruk dwustronny. Jednocześnie akceptuje ona dowolne oprogramowanie dla wcześniejszych drukarek z rodziny LaserJet II, zużywając dwukrotnie mniej papieru.

LaserJet II D ma dwa zasobniki na 200 arkuszy każdy i może być wyposażona w dodatkowy podajnik kopert (za 350 dolarów). Wszystkie operacje podawania papieru są wykonywane z klawiatury.

Tak jak w poprzednich drukarkach z tej rodziny zastosowano w niej mechanizm drukujący Canon SX, zapewniający rozdzielczość 300 punktów na cal. Szybkość drukowania pozostała taka sama – 8 stron na minutę. Wiele elementów wyglądu zewnętrznego jest identycznych, z tym że wysokość jest dwukrotnie większa, a waga wzrosła z 22,7 do 33,6 kg w stosunku do LaserJet II.

Po wydrukowaniu jednej powierzchni arkusza, drukarka wyrzuca arkusz do tylnego zasobnika, a następnie podaje go do wydruku odwrotną stroną. Według przeprowadzonych badań uzyskuje się poprawną pracę aż do 11 kg papieru zadrukowanego. Proces wydruku jest tylko 10% dłuższy niż przy pisaniu jednostronnym. Można wybierać bok, wzdłuż którego mają być składane dwustronnie zapisane strony. Na razie dokonuje się tego na płycie czołowej, ale opracowywane obecnie programy sterujące drukarką (ang. *drivers*) umożliwią drukowanie dwustronne we wszystkich znanych systemach przetwarzania tekstów. Już obecnie można korzystać z edytora Word Perfect 5.0.

Dodatkowy podajnik na 50 kopert jest usytuowany nad górnym zasobnikiem i jest zasilany z gniazda z boku drukarki. Bez tego urządzenia można podawać koperty ręcznie lub korzystać z podajnika na 15 kopert (w cenie 85 dolarów) stosowanego przy dawnych drukarkach. Nowy podajnik jest wygodniejszy i bardziej niezawodny.

Są pewne rozszerzenia zestawów czcionek w stosunku do poprzednich typów, przy czym wszystkie kasety stosowane poprzednio mogą tu być użyte. Drukarka ta automatycznie odwraca położenie czcionek z poziomego na pionowe, co eliminuje potrzebę przechowywania czcionek pionowych, lecz wymaga większej pojemności pamięci.

LaserJet II D jest wyposażony w łączówkę alternatywnego sprzęgu wejścia-wyjścia, lecz nie obejmuje to sprzęgu wizyjnego, stosowanego w Series II do dołączania pakietów PostScript.

Sieniawski L.: Architektury zredukowane

INFORMATYKA 1990, nr 5, s. 1

Geneza komputerów o architekturze zredukowanej oraz charakterystyka wybranych realizacji systemów o architekturze RISC, zawierająca omówienie wniosków z eksperymentów nad ulepszeniem architektury komputerów, a także historię zmian poglądów na zasady konstruowania sprzętu w minionym dwudziestoleciu.

Sieniawski L.: RISC architectures

INFORMATYKA 1990 No. 5, p. 1

Genesis of RISC and characteristics of selected RISC realizations, which includes discussion of conclusions from research experiments on improvement of computer architecture, as well as the history of computer design philosophy during last 20 years.

Sieniawski J.: RISC-Architekturen

INFORMATYKA 1990, Nr 5, S. 1

Genese und eine Charakteristik der ausgewählten RISC--Realisationen, die eine Schlussfolgerungenbesprechung von Forschungsexperimenten im Bereich der Computerarchitekturrationalisierung, sowie eine Geschichte von Meinungen über Konstruierungsgrundsätze der Hardware in vorigen 20 Jahren, umfasst.

Malec J.: Maszyny lispowe (1)

INFORMATYKA 1990, nr 5, s. 7

Pierwsza część charakterystyki maszyn lispowych, zawierająca omówienie prototypów tego rodzaju komputerów.

Malec J.: Lisp machines (1)

INFORMATYKA 1990, No. 5, p. 7

First part of characteristics of the Lisp machines, which includes discussion of prototypes constructed in this area.

Malec J.: Lisp-Maschinen (1)

INFORMATYKA 1990, Nr. 5, S. 7

Erster Teil einer Charakteristik von Lisp-Maschinen, der eine Besprechung von gebauten Prototypen dieser Computerkategorie umfasst.

Dembiński P.: Protokoły komunikacyjne (1)

INFORMATYKA 1990, nr 5, s. 13

Pierwsza część poglądowego omówienia istoty oraz funkcji protokołów w sieciach komputerowych o architekturze warstwowej.

Dembiński P.: Communication protocols (1)

INFORMATYKA 1990, No. 5, p. 13

First part of the demonstrative discussion of protocol essence and functions in computer networks with layer architecture.

Dembiński P.: Kommunikationsprotokolle (1)

INFORMATYKA 1990, Nr. 5, S. 13

Erster Teil einer anschaulichen Besprechung des Wesens und Funktionen von Protokollen in Computernetzen mit Schichtarchitektur.

Żakowicz T.: CDS/ISIS – narzędzie do zarządzania bazami danych (1)

INFORMATYKA 1990, nr 5, s. 16

Pierwsza część charakterystyki najnowszej wersji pakietu programów CDS/ISIS, przeznaczonego do tworzenia i operowania bazami danych na dużych komputerach IBM.

Żakowicz T.: CDS/ISIS – a tool for data base management (1)

INFORMATYKA 1990, No. 5, p. 16

First part of characteristics of the most recent version of the CDS/ISIS program package devoted for data base building and management on IBM main frame computers.

Żakowicz T.: CDS/ISIS – ein Werkzeug zur Datenbankverwaltung (1)

INFORMATYKA 1990, Nr. 5, S. 16

Erster Teil einer Charakteristik von neuester Version des CDS/ISIS–Programmpakets, das zur Datenbankengestalltung und -Handlung auf IBM-Grossrechnern bestimmt ist.

Bronowska M.: System graficzny Ramtek RM-9460

INFORMATYKA 1990, nr 5, s. 21

Szczegółowa charakterystyka konstrukcji, oprogramowania i możliwości funkcjonalnych systemu graficznego Ramtek RM-9460, współpracującego z minikomputerami dużej mocy typu VAX 750.

Bronowska M.: Ramtek RM-9460 graphic display system

INFORMATYKA 1990, No. 5, p. 21

Detailed characteristics of the RM-9460 graphic display system hardware, software and functional possibilities, which cooperates with big minicomputers of the Vax 750 category.

Bronowska M.: Ramtek RM-9460-Graphiksystem

INFORMATYKA 1990 Nr. 5, S. 21

Eine detaillierte Charakteristik von Hardware, Software und funktionellen Möglichkeiten des RM-9460-Graphiksystems, das mit grossen Minirechnern von VAX 750-Kategorie arbeitet.

Runka H. J.: Duża macierz w Turbo Pascalu 5.0

INFORMATYKA 1990, nr 5, s. 23

Charakterystyka rozwiązania pozwalającego rozszerzyć możliwości implementacji algorytmów w języku Turbo Pascal 5.0.

Runka H. J.: Big matrix in the Turbo Pascal 5.0

INFORMATYKA 1990, No. 5, p. 23

Characteristics of the solution which extends possibilities of algorithm implementation in the Turbo Pascal 5.0 language.

Runka H. J.: Grossmatrix in Turbo Pascal 5.0

INFORMATYKA 1990 Nr. 5, S. 23

Eine Charakteristik von der Lösung, die eine Erweiterung von Möglichkeiten der Algorithmenimplementierung in Turbo Pascal 5.0-Sprache erlaubt.

Odpowiedzialność za jakość wyrobów

Przebudowa naszego systemu gospodarczego na model rynkowy i pełne otwarcie na świat wymaga generalnej zmiany podejścia w każdej dziedzinie działalności. Szczególnie ważną, a jednocześnie chyba najbardziej zdeformowaną dziedziną jest kompleks zagadnień związanych z odpowiedzialnością za jakość wyrobów. Przez dziesięciolecia ukształtował się u nas model przytłaczającej dominacji producenta nad użytkownikiem. W sytuacji permanentnego kurczenia się oferty rynkowej użytkownik został zepchnięty na pozycję petenta, dla którego szczytem szczęścia było „zdobycie" wyrobu odrzuconego przez odbiorcę zagranicznego w wyniku oczywistych wad tego wyrobu.

W wyjątkowo trudnej sytuacji był użytkownik sprzętu komputerowego i oprogramowania, popadający od przeszło 20. lat w ciągle wzrastającą zależność od krajowego monopolisty, któremu udało się podporządkować również decyzje importowe. W przypadku złej jakości wyrobów konsumpcyjnych (tzw. rynkowych) z producentami-monopolistami walczyła w ostatnim dziesięcioleciu, choć z niewielkim skutkiem, Federacja Konsumentów. Użytkownicy wyrobów tzw. pozarynkowych, do których należał sprzęt komputerowy i jego oprogramowanie, niestety nie mieli tego rodzaju instytucjonalnej obrony swych interesów. Krytyczne wypowiedzi i apele, jakie starało się rozpowszechniać Polskie Towarzystwo Informatyczne, nie spowodowały odczuwalnej zmiany podejścia monopolisty, a w konsekwencji nie doprowadziły do poprawy sytuacji krajowego użytkownika. Niewłaściwe, a często również mało odpowiedzialne podejście do użytkownika spotkać można ostatnio niestety nie tylko ze strony zbiurokratyzowanych przedsiębiorstw państwowych, ale również niektórych firm prywatnych. Wykorzystując znaczny popyt na sprzęt komputerowy oraz często ograniczoną wiedzę informatyczną użytkownika, a także kusząc nabywców bardzo niskimi – w porównaniu do ofert renomowanych firm – cenami sprzętu, sprzedają one często wyroby wątpliwej jakości.

Zjawisko takie pojawiło się w olbrzymiej skali również na Zachodzie z chwilą masowego wejścia na rynek drobnych sprzedawców sprzętu mikrokomputerowego produkcji wschodnioazjatyckiej. Było to prawdopodobnie jedną z przyczyn przyspieszenia opracowania dla wszystkich krajów EWG jednolitych przepisów prawnych, dotyczących odpowiedzialności za jakość wyrobów. Przepisy te weszły w życie na początku br. w postaci ustawy i należy je rozumieć jako jeden z istotnych elementów przygotowań organizacyjnych do tak bliskiego już utworzenia Zjednoczonej Europy. W zbliżającej się perspektywie pełnego otwarcia naszej gospodarki na świat celowe jest zapoznanie się z głównymi postanowieniami wspomnianej ustawy, które wprawdzie w najbliższym czasie ugodzą dotkliwie w naszych eksporterów, ale z drugiej strony przyniosą odczuwalne korzyści użytkownikom importowanego sprzętu i to nie tylko komputerowego.

Wspomniana ustawa EWG oznacza zwiększenie skuteczności ochrony użytkownika przed szkodami, jakie mogą dla niego wyniknąć wskutek ukrytych wad wyrobu. Dla producentów i sprzedawców oznacza zwiększenie ich odpowiedzialności z tytułu dostarczanych produktów. Producent będzie odpowiadał za wszelkie szkody, jakie wynikły z udowodnionych wad jego produktu i jest zobowiązany do wypłacenia z tego tytułu użytkownikowi odpowiedniego odszkodowania. Producentem w rozumieniu ustawy jest również ten, który bezpośrednio wyrobów nie wytwarza, ale umieszcza na nich swój znak firmowy. Sprzedawca wyrobu (dystrybutor, hurtownik, detalista) jest traktowany także jako producent wówczas, gdy rzeczywistym wytwórcą jest firma pozaeuropejska bez oficjalnego przedstawicielstwa w Europie lub gdy wytwórcy tego nie można w sposób jednoznaczny zidentyfikować w ciągu jednego miesiąca od momentu wystąpienia szkody. Odpowiedzialność za szkody obejmuje okres 10. lat od chwili wprowadzenia wyrobu do eksploatacji, natomiast poszkodowany użytkownik musi swoje pretensje uzasadnić najpóźniej w ciągu trzech lat od chwili wystąpienia szkody. Odpowiedzialność finansowa w przypadku zgonu lub uszkodzenia cielesnego użytkownika jest ograniczona do wysokości ok. 100 mln dol. USA, natomiast dla szkód rzeczowych górnej granicy odszkodowania nie określono. Ustawa przewiduje bardzo ograniczony zakres przypadków stwarzających dla producenta możliwość uchylenia się od odpowiedzialności za wyrządzoną szkodę.

Ze zrozumiałych względów istotny wpływ na eliminację wad wyrobu może mieć wyłącznie jego producent. Z faktu tego wynika olbrzymia odpowiedzialność dystrybutorów produktów OEM, którzy stanowią znaczącą część dostawców sprzętu komputerowego. Chociaż z reguły nie mają oni żadnego lub tylko niewielki wpływ na jakość odbieranych od producentów wyrobów, to jednak firmując sprzedawany sprzęt ponoszą pełną odpowiedzialność za ujawnione przez użytkownika wady produktu. W znacznie lepszej sytuacji jest typowy sprzedawca sprzętu komputerowego, który skutecznie może bronić się przed roszczeniami użytkownika odsyłając go po prostu do konkretnego producenta wyrobu. Oczywiście nie ma on takiej możliwości, jeżeli sprzedaje anonimowe wyroby importowane z Dalekiego Wschodu.

Wzrost odpowiedzialności za dostarczony wyrób spowoduje, że zarówno producent, jak i sprzedawca będą zwracać większą uwagę na jednoznaczność sformułowań oraz kompletność instrukcji obsługi wyrobu. W instrukcjach tych powinno być szczególnie wyraźnie wyeksponowane ryzyko nieprawidłowej obsługi wyrobu. Nowa ustawa powinna więc doprowadzić do zasadniczego przewartościowania dotychczasowego podejścia producenta, który najchętniej przerzucał odpowiedzialność za szkody na użytkownika, zakładając, że były one rezultatem błędów w obsłudze wyrobu.

Postanowienia nowej ustawy stały się w krajach EWG już obowiązujące i dlatego nie można się już tam powoływać na znacznie liberalniejsze dla producentów i sprzedawców ogólne przepisy prawne regulujące sferę umów i dostaw wyrobów. A na marginesie tej ustawy warto zauważyć, że widoczne są w niej protekcjonistyczne intencje obrony interesów Wspólnoty Europejskiej przed zalewem tanich wyrobów z Dalekiego Wschodu, które skutkiem nowych przepisów staną się mniej atrakcyjne dla tak licznych obecnie sprzedawców tych wyrobów.

(WK)