4 1989

informatyka

Cobol [illegible] PC
Układy szybkiego [illegible] mnożenia
Rekonstruowanie obrazów

WYDAWNICTWO NOT SIGMA

Cena 300 zł

informatyka

4 1989

Nr 4

Miesięcznik Rok XXIV
Kwiecień 1989

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Redakcja: 01-517 Warszawa, ul. Mickiewicza 18
m. 17, tel. 39-14-34

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13, 85-950 BYDGOSZCZ
Zam. 555/89 E 10
Obj. 4,0 ark. druk. Nakład 8500 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 300 zł
Prenumerata roczna 3600 zł

WYDAWNICTWO
NACZELNA ORGANIZACJA TECHNICZNA
CZASOPISM I KSIĄŻEK TECHNICZNYCH
SIGMA
00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE: strona

W NAJBLIŻSZYCH NUMERACH:

- Wacław Iszkowski dokonuje przeglądu metod zwiększenia szybkości działania komputerów.
- Jarosław Deminet omawia badania przepustowości sieci lokalnych Transnet, Ethernet i Arcnet.
- Piotr Fuglewicz i Tadeusz Korniak prezentują trzy różne systemy baz danych dla rodziny Mera 600: Podsystem Aktualizacji Kartotek, ReDS i mBase.
- Jacek Skrzymowski zaczyna cykl publikacji na temat nowej normy języka Fortran.
- Kazimierz Kaczmarczyk charakteryzuje moduł procesora mikrokomputerów rodziny Mera 600.
- Wiesław Nowiński opisuje rekonstruowanie obrazów metodami rozwinięcia w szereg.

MAREK PAWŁOWSKI
Instytut Informatyki
Politechnika Warszawska

Przegląd układów szybkiego mnożenia

Rozwój zastosowań cyfrowego przetwarzania informacji spowodował, że coraz więcej skomplikowanych działań, dotychczas wykonywanych programowo, musi być zrealizowanych sprzętowo. Główną przyczyną takiego postępowania była i jest konieczność minimalizacji czasu wykonywania tych obliczeń oraz minimalizacja wielkości urządzeń, które mają realizować swoje zadania – na przykład w warunkach pokładowych samolotu.

Radar jako pierwszy wymusił rozwój cyfrowego przetwarzania sygnałów, opartego m. in. na szybkiej realizacji mnożenia. Konieczna jest ona również przy przetwarzaniu obrazów, a zwłaszcza podczas ich animowania.

W obu wypadkach najczęściej wykonywaną złożoną operacją arytmetyczną jest mnożenie kilkunastobitowych argumentów. Realizacja tej operacji przy użyciu uniwersalnych mikroprocesorów 16-bitowych (np. 8086, MC68000) wymaga od kilku do kilkudziesięciu μs. Przy dużej liczbie przetwarzanych danych czas takiego mnożenia jest zbyt długi. Musiały więc powstać specjalne układy scalone do szybkiego wykonywania operacji mnożenia.

SZYBKIE UKŁADY MNOŻĄCE

Duża rodzina układów scalonych umożliwia kombinacyjne wykonanie operacji mnożenia dwóch n-bitowych danych reprezentujących liczby w kodzie *U*2 lub *NKB* albo *U*2 i *NKB*. W wyniku mnożenia otrzymuje się słowo $2n$-bitowe. W zależności od typu układu liczba n jest równa 8, 12, 16 lub 32. Maksymalny czas mnożenia wynosi od 40 do 200 ns.

Najbardziej liczna jest rodzina układów mnożących produkowanych przez firmę TRW. Zestawienie tych układów przedstawiono w tabeli.

Układy mnożące firmy TRW

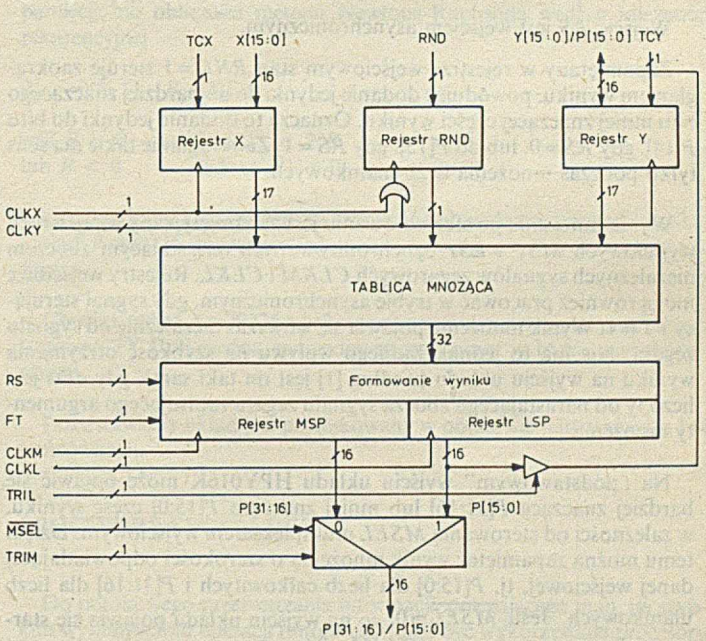
Oznaczenie	Długość argumentu wejściowego	Długość wyniku	Czas mnożenia (ns)	Kod argumentu
MPY008H	8	16	90	*U*2
MPY008H-1	8	16	65	*U*2
MPY08HU	8	16	90	*NKB*
MPY08HU-1	8	16	65	*NKB*
MPY012H	12		115	*U*2/*NKB*
MPY112K	12	16	50	*U*2/*NKB*
MPY016H	16	32	145	*U*2/*NKB*, *U*2 i *NKB*
MPY016K	16	32	45	*U*2/*NKB*, *U*2 i *NKB*
MPY016K-1	16	32	40	*U*2/*NKB*, *U*2 i *NKB*

Wszystkie te układy mają podobną strukturę wewnętrzną, w której można wyróżnić następujące bloki funkcjonalne:

- blok układów wejściowych, zawierający dwa rejestry zapamiętujące n-bitowe dane wejściowe, synchronizowane niezależnymi sygnałami zegarowymi, oraz przerzutniki przechowujące informacje o kodzie danych wejściowych, podawane równocześnie z argumentami mnożenia;
- blok tablicy mnożącej, będący układem kombinacyjnym zbudowanym z tablicy sumatorów generujących n wyników częściowych; wynik mnożenia jest zaokrąglany i formatowany,
- blok układów wyjściowych, zawierający rejestr wyniku oraz bufor trójstanowy; wynik najczęściej składa się z dwóch n-bitowych słów zapamiętanych w dwóch niezależnie synchronizowanych rejestrach; słowa te mogą pojawić się na różnych wyjściach układu lub za pośrednictwem multipleksera na jednym z nich.

Jak wynika z tabeli, argumentami mnożenia mogą być słowa n-bitowe reprezentujące liczby ułamkowe lub całkowite w kodzie *U*2 lub *NKB*.

Z wymienionych układów najbardziej rozbudowany jest układ MPY016K (rys. 1). Umożliwia on mnożenie liczb ułamkowych lub całkowitych reprezentowanych 16-bitowymi słowami *X* i *Y*, zapamiętanymi w rejestrach wejściowych układu. Dla układu nie ma żadnych różnic między mnożeniem liczb całkowitych a mnożeniem liczb ułamkowych; o tym, które mnożenie zostało wykonane, decyduje interpretacja jego wyniku. Po mnożeniu liczb całkowitych „podstawową" częścią wyniku jest zawartość *P*[15:0], a po mnożeniu liczb ułamkowych zasadniczą częścią wyniku jest zawartość *P*[31:16] oraz zawartość *P*[15:0] zwiększająca precyzję obliczeń.



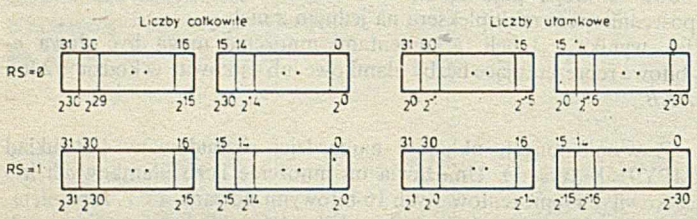
Rys. 1. Schemat blokowy układu MPY016K

Argumenty mnożenia są zapamiętywane w rejestrach wejściowych synchronizowanych narastającym zboczem niezależnych sygnałów zegarowych *CLKX* i *CLKY*. Wynika stąd, że dane wejściowe mogą być podawane w różnych momentach, w szczególności mogą być brane z tego samego źródła, gdy *X* i *Y* są ze sobą połączone. Fakt niezależnego synchronizowania rejestrów wejściowych może być również wykorzystany do realizacji operacji mnożenia ciągu liczb przez stałą, która może być zapamiętana w jednym z nich na początku całej operacji.

Równocześnie z argumentami mnożenia do rejestrów wejściowych jest zapisywana informacja o kodzie, w jakim są reprezentowane dane wejściowe. Wejścia *TCX* i *TCY* niezależnie od siebie decydują o sposobie interpretacji słów *X* i *Y*. Gdy *TCX* lub *TCY* jest logiczną jedynką, to *X* lub *Y* reprezentuje liczbę w kodzie *U*2, a gdy jest zerem, to dana wejściowa jest w kodzie *NKB*. Stan wejść *TCX* i *TCY* może być dowolny, a więc jest możliwe wykonywanie operacji mnożenia na argumentach w różnych kodach. Można np. mnożyć liczbę reprezentowaną w kodzie *NKB* (bez znaku) przez liczbę reprezentowaną w kodzie *U*2 (ze znakiem), otrzymując w wyniku liczbę w kodzie *U*2.

Podczas zapisywania słowa *X* lub *Y* do rejestru wejściowego zapisywany jest również stan wejścia *RND*, decydujący o sposobie zaokrąglania wyniku mnożenia. Wpływ *RND* na wynik mnożenia jest uzależniony od stanu wejścia *RS* sterującego formatowaniem danych wyjściowych.

Stan wejścia sterującego *RS* ma znaczenie podczas mnożenia liczb ze znakiem, tzn. w kodzie *U2*. Jeśli *RS*=0, to znak wyniku mnożenia jest umieszczany na pozycjach 31 i 15 w 32-bitowym słowie wyniku. Oznacza to, że również młodsza część wyniku zawiera bit znaku. Pozycja 31 wyniku ma wagę 2^{30} przy mnożeniu liczb całkowitych, a 2^{0} przy mnożeniu liczb ułamkowych. Jeśli *RS*=1, to bit *P*[15] ma wagę 2^{15} dla liczb całkowitych, a 2^{-15} dla ułamkowych, natomiast bit znaku *P*[31] wagę odpowiednio 2^{31} i 2^{1} (rys. 2).



Rys. 2. Waga pozycji mnożenia P [31:0] dla argumentów w kodzie U_2 i różnych stanów wejścia RS

Przy mnożeniu liczb reprezentowanych w kodzie *NKB* lub *NKB* i *U2* stan wejścia *RS* musi być równy jedynce logicznej, aby uzyskany wynik był prawidłowy.

Wejście *RS* jest wejściem asynchronicznym.

Zapamiętany w rejestrze wejściowym stan *RND*=1 steruje zaokrąglaniem wyniku, powodując dodanie jedynki do najbardziej znaczącego bitu mniej znaczącej części wyniku. Oznacza to dodanie jedynki do bitu *P*[14], gdy *RS*=0, lub do *P*[15], gdy *RS*=1. Zaokrąglanie takie ma sens tylko podczas mnożenia liczb ułamkowych.

Wynik mnożenia po sformatowaniu jest zapamiętywany w rejestrach wyjściowych *MSP* i *LSP* synchronizowanych narastającym zboczem niezależnych sygnałów zegarowych *CLKM* i *CLKL*. Rejestry wejściowe mogą również pracować w trybie asynchronicznym, gdy sygnał sterujący *FT*=1. Wynik mnożenia pojawia się wówczas niezależnie od sygnału zegara. Nie ma to jednak żadnego wpływu na szybkość otrzymania wyniku na wyjściu układu (według [1] jest on taki sam), gdy czas jest liczony od narastającego zbocza sygnału zegara zapisującego argumenty mnożenia.

Na „podstawowym" wyjściu układu HPY016K może pojawić się bardziej znacząca *P*[31:16] lub mniej znacząca *P*[15:0] część wyniku, w zależności od sterowania *MSEL* multiplekserem wyjściowym. Dzięki temu można zapamiętać wynik mnożenia o szerokości odpowiadającej danej wejściowej, tj. *P*[15:0] dla liczb całkowitych i *P*[31:16] dla liczb ułamkowych. Jeśli $\overline{MSEL}$=0, to na wyjściu układu pojawia się starsza część wyniku, gdy sygnał *TRIM*=0 (w przeciwnym razie wyjście jest w stanie dużej impedancji). Mniej znacząca część wyniku może być wtedy odczytana na liniach *Y*, gdy *TRIL*=0.

Niezależne sygnały synchronizujące działanie rejestrów wewnętrznych układu MPY016K umożliwiają stosowanie zegara wielofazowego i tym samym zwiększenie szybkości działania układu. Czas mnożenia w trybie synchronicznym, tzn. gdy wynik jest zapamiętywany w rejestrach wyjściowych, wynosi 40–45 ns (w zależności od wersji układu) i oznacza minimalny odstęp czasu między narastającymi zboczami sygnałów synchronizujących zapamiętanie danych do mnożenia i wyniku. Czas propagacji wyniku przez rejestry wyjściowe, multiplekser i bufor trójstanowy jest równy maksymalnie 30 ns. A więc „prawdziwy" czas mnożenia wynosi maksymalnie 70–75 ns i jest taki sam, jak w trybie asynchronicznym.

Układ MPY016K jest wytwarzany w obudowie *DIL*64 i zasilany pojedynczym napięciem +5V.

Odpowiednikiem funkcjonalnym układu MPY016K jest układ Am 29516 [2], o strukturze wewnętrznej zrealizowanej w technologii ECL. Podobną architekturę ma układ Am29517, w którym rejestry wewnętrzne są synchronizowane jednym sygnałem zegarowym. Odpowiednikiem niezależnych zegarów układu MPY016K są sygnały sterujące zapisem do poszczególnych rejestrów.

Odpowiednikiem układu Am29517 jest LMU18 wykonany w technologii CMOS [3]. Charakteryzuje się on jednoczesnym dostępem do całego wyniku, stąd obudowa matrycowa 84-końcówkowa. Czas cyklu wynosi maksymalnie 80 ns. W ZSRR produkuje się odpowiednik funkcjonalny układu MPY016K oznaczony symbolem KM1802BP5.

Rozbudowaną wersją układów mnożących są układy scalone dodatkowo wyposażone w możliwość dodawania (lub odejmowania) aktualnego wyniku mnożenia do (od) zawartości rejestru wyjściowego nazywanego akumulatorem. Operacja dodawania lub odejmowania jest wykonywana w odpowiednio rozbudowanej tablicy mnożącej, zbudowanej z bramek *AND* i sumatorów. Dwa wejścia sterujące *ACC* i *SUB* determinują wykonanie operacji na poprzedniej zawartości akumulatora (*ACC*) oraz dodania lub odjęcia (*SUB*).

Rejestr akumulatora jest 35-bitowy, przy czym trzy najbardziej znaczące bity są dostępne na specjalnym wyjściu układu. Rejestr akumulatora można wstępnie zapisać, czym steruje sygnał *PREL*.

Firma TRW oferuje pięć układów mnożących wyposażonych w rejestr akumulatora. Czas mnożenia w tych układach jest równy od 100 do 165 ns w zależności od typu układu. Układ TDC1008 umożliwia mnożenie argumentów 8-bitowych, TDC1009 – słów 2-bitowych, a układy TDC1010, TDC1043 i TMC2010 – 16-bitowych. Układ TDC1043 umożliwia odczytanie tylko bardziej znaczącej części wyniku. Odpowiednikiem funkcjonalnym układu TDC1010 jest układ Am29510.

W trakcie wykonywania mnożenia mogą pojawić się błędy przemijające, zmieniające wynik całego ciągu obliczeń. Opisane wcześniej układy nie ułatwiają uniknięcia tego rodzaju błędów, nie umożliwiają także ich łatwego wykrywania. Pewną próbą zwiększenia niezawodności działania układów mnożących jest układ Am29323, mnożący dane 32-bitowe. Ma on dodatkowo wbudowane komparatory umożliwiające porównanie stanu wyjść ze stanem wyjść drugiego układu tego samego typu, realizującego te same operacje.

Każdy bajt danych wejściowych i wyniku ma swój bit parzystości. Układ Am29323 sprawdza parzystość danych wejściowych, generując sygnał błędu parzystości, oraz generuje 4 bity *PP* parzystości wyniku. Układ może pracować w trybie *MASTER* lub *SLAVE* (gdy sygnał sterujący *SLAVE*=1). W trybie *SLAVE* układ śledzi pracę bliźniaczego układu pracującego w trybie *MASTER*, przez porównywanie stanu połączonych ze sobą odpowiednich linii wyjściowych. Różnica między stanem linii sterowanych przez układ *MASTER* a stanem wewnętrznym układu *SLAVE* (nieaktywne bufory wyjściowe) powoduje wygenerowanie sygnału błędu *HARDERR*.

Układ sterujący działaniem układu mnożącego, badając stan linii *HARDERR* i *PARERR*, może w wypadku sygnalizowanego błędu powtórzyć ostatnio wykonywaną operację.

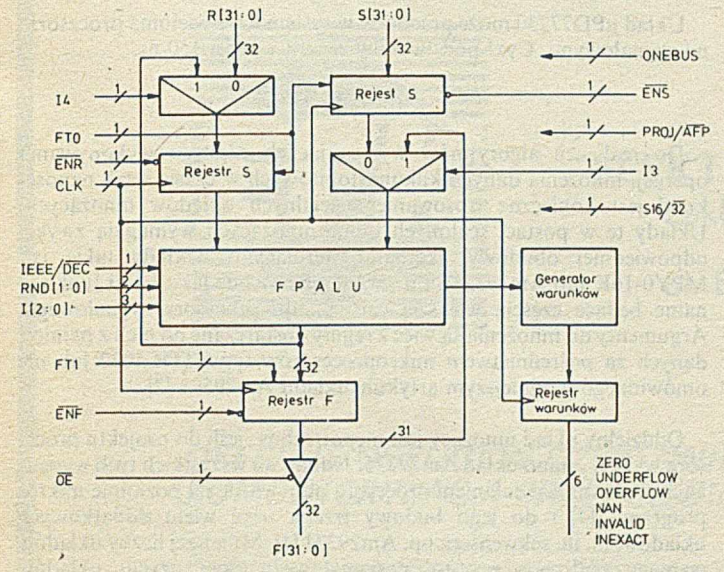
ZMIENNOPRZECINKOWE BLOKI ARYTMETYCZNE

W wielu zastosowaniach szybka realizacja operacji stałoprzecinkowych nie wystarcza do przetwarzania liczb z dużego zakresu. Konieczne jest wtedy wykonywanie operacji na liczbach zmiennoprzecinkowych. Operacje zmiennoprzecinkowe wymagają jednoczesnego działania na liczbach ułamkowych (mantysa) i liczbach całkowitych (cecha). Sam układ mnożący nie wystarcza więc do realizacji mnożenia zmiennoprzecinkowego, konieczny jest jeszcze sumator dla cech, układ normalizujący wynik oraz układ sygnalizujący sytuacje wyjątkowe (nadmiar, niedomiar itp.). Funkcje tych bloków spełnia zmiennoprzecinkowa jednostka arytmetyczno-logiczna będąca częścią układu Am29325 [3].

Układ Am29325 umożliwia dodawanie, odejmowanie i mnożenie 32-bitowych argumentów zmiennoprzecinkowych w jednym cyklu zegara (100 ns). Akceptuje dane w formacie *IEEE* (*P*754) i *DEC*, przy czym jest zapewniona konwersja danych z jednego formatu na drugi. Struktura wewnętrzna układu jest wykonana w technologii ECL, ale układ zasilany jest pojedynczym napięciem +5V (rys. 3).

Działanie zmiennoprzecinkowej jednostki arytmetyczno-logicznej *FPALU* można podzielić na trzy etapy:

– wykonanie operacji wybranej przez sterowanie I [2:0], z wynikiem o maksymalnej precyzji,
– zaokrąglanie wyniku do postaci 32-bitowej zgodnie ze sterowaniem RND [1:0],
– akceptacja wyniku – wygenerowanie warunków:
ZERO – w wyniku otrzymano zero,
UNDERFLOW – niedomiar,
OVERFLOW – nadmiar,
NAN – w wyniku otrzymano nie-liczbę,
INVALID – wykonano nieprawidłową operację (np. dzielenie przez zero),
INEXACT – wynik nie jest maksymalnie dokładny.

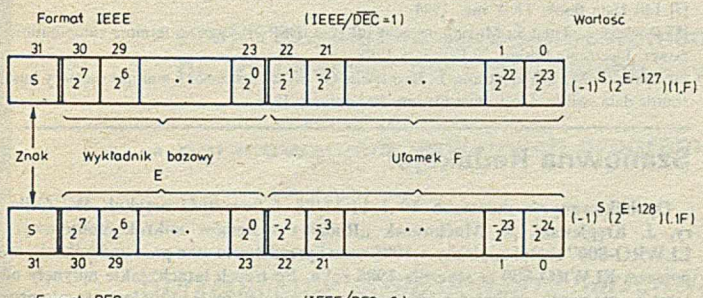


Rys. 3. Schemat blokowy układu Am29325

Blok *FPALU* umożliwia wykonanie jednej z ośmiu operacji na liczbach zmiennoprzecinkowych w formacie *IEEE* lub *DEC* w zależności od sterowania IEEE/$\overline{\text{DEC}}$

– dodanie $R+S$
– odjęcie $R-S$,

odjęcie $2-S$ (operacja przydatna podczas wykonywania operacji szybkiego dzielenia metodą Newtona-Raphsona),
– mnożenie $R \times S$,
– konwersja liczby całkowitej na zmiennoprzecinkową,
– konwersja liczby zmiennoprzecinkowej na całkowitą,
– konwersja danej z formatu *IEEE* na daną w formacie *DEC*,
– konwersja danej z formatu *DEC* na daną w formacie *IEEE*.
Formaty danych wejściowych i wyjściowych do (z) układu Am29323 przedstawiono na rys. 4.



Rys. 4. Formaty danych dla układu Am29325

Układ Am29325 przetwarza tylko liczby zmiennoprzecinkowe znormalizowane i takie też wytwarza. Liczba nieznormalizowana jest zamieniana na 0 z zachowaniem znaku.

W trzecim etapie działania *FPALU* mogą być wykryte m.in. następujące sytuacje wyjątkowe.

- Występuje nadmiar dodatni lub ujemny; jest wtedy ustawiany warunek *OVERFLOW*=1, a wynik operacji przyjmuje wartość $+\infty$ przy nadmiarze dodatnim i $-\infty$ przy nadmiarze ujemnym bez względu na sposób zaokrąglania.
- Moduł wyniku operacji jest mniejszy od wartości 2^{-126} i wynik nie jest równy precyzyjnej nieskończoności; jest wtedy ustawiany warunek *UNDERFLOW*=1.
- Otrzymano wynik, którego nie można znormalizować; są wtedy ustawiane warunki *INEXACT*=1, *UNDERFLOW*=1 i *ZERO*=1, a wynik jest zmieniany na 0.

Nieprawidłowe operacje, np. dzielenie przez zero, mnożenie zera przez nieskończoność, dodawanie $+\infty$ do $-\infty$, są sygnalizowane przez ustawienie warunku *INVALID*=1, a wynik jest zmieniany na nie-liczbę o postaci $E=255$, B [22]=0, $S=0$ i dowolnymi pozostałymi bitami ułamka. Pojawienie się takiej nie-liczby jako danej wejściowej powoduje przeniesienie jej na wyjście bez zmian i ustawienie warunku *NAN*=1.

Wynik operacji w *FPALU* jest dłuższy od wyprowadzanego, dlatego musi on być zaokrąglony. Sposób zaokrąglania zależy od sterowania *RND*[1:0]:
– RND [1:0]=00 – zaokrąglenie do wartości najbliższej,
– RND [1:0]=01 – zaokrąglenie do $-\infty$,
– RND [1:0]=10 – zaokrąglenie do $+\infty$,
– RND [1:0]=11 – zaokrąglenie do 0.

Układ Am29325 nie zapewnia wprost realizacji funkcji dzielenia liczb zmiennoprzecinkowych, można je jednak wykonać za pomocą mnożenia dzielnej przez odwrotność dzielnika, tzn. $C=A/B \equiv C=A * 1/B$. Odwrotność dzielnika może być odczytana z pamięci stałej umieszczonej na wejściu R układu Am29325 (wadą tego rozwiązania jest duża pojemność tej pamięci) lub obliczona metodą Newtona-Raphsona według zależności rekurencyjnej

$$X_{i-1} = X_i (2 - B * X_i)$$

jeśli $B > 0$ $\quad 0 < X_0 < 2/B$
lub $B < 0$ $\quad 2/B < X_0 < 0$

w której po kilku krokach można otrzymać zadowalającą precyzję wyniku. Układ Am29325 zapewnia wykonanie operacji $2-B$ w jednym cyklu zegara.

Chociaż układ Am29325 zawiera rejestry wewnętrzne, to przy użyciu sterowań *FT* [1:0] można spowodować jego pracę w trybie asynchronicznym.

Przedstawiony układ jest produkowany w obudowie matrycowej o 144 końcówkach.

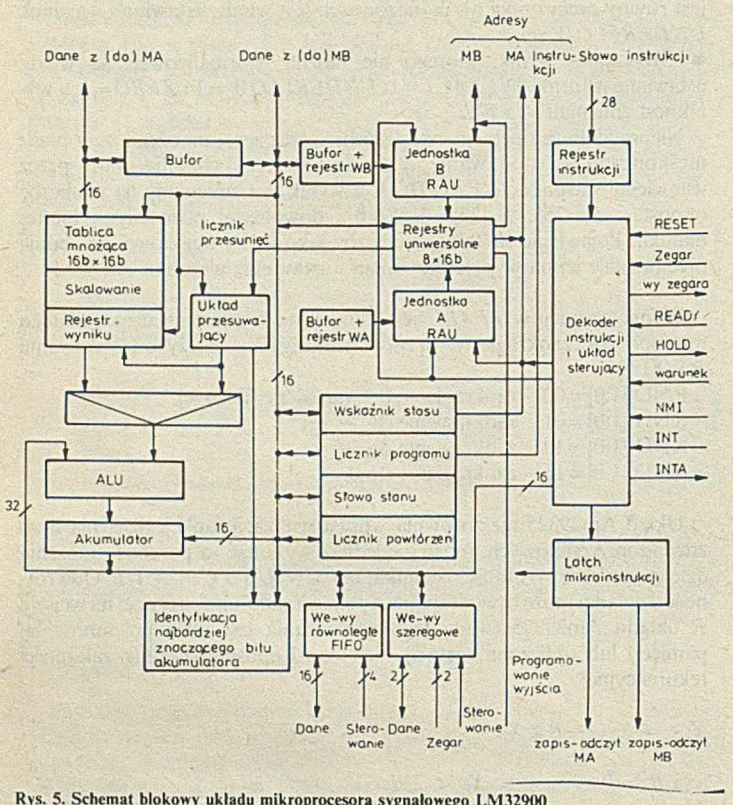
PROCESORY SYGNAŁOWE

Do potokowego przetwarzania informacji, polegającego m.in. na szybkim wykonywaniu operacji mnożenia, mogą być wykorzystane procesory sygnałowe. Najczęściej układy scalone procesorów sygnałowych współpracują z zewnętrznymi pamięciami programu i danych. Zwykle występują dwie pamięci danych, jednocześnie dostarczające argumenty operacji (co przyspiesza obliczenia). Dlatego układy te współpracują z otoczeniem za pośrednictwem wielu szyn. Przykładem takiego rozwiązania jest układ mikroprocesora LM32900 [9].

Układ LM32900 (rys. 5) generuje trzy adresy w każdym cyklu: jeden dla 28-bitowej pamięci programu i dwa adresy dla dwóch 16-bitowych pamięci danych. Dane odczytane z obu pamięci danych *MA* i *MB* mogą być argumentami mnożenia w jednym cyklu działania układu. Trzy poziomy przetwarzania potokowego powodują nałożenie fazy pobrania rozkazu z fazami jego dekodowania i wykonania.

Adresy danych są obliczane w dwóch układach rejestrów *RAU* umożliwiających wyznaczenie następnika lub poprzednika adresu lub pamięci *MA* i *MB*. Układ mnożący akceptuje dane w kodzie *NKB* i *U2* w dowolnej kombinacji, *ALU* natomiast jest jednostką arytmetyczną działającą na liczbach w kodzie *U2*. Układ LM32900 współpracuje z procesorem nadrzędnym za pośrednictwem dwóch buforów *FIFO* (16×16), wejściowego i wyjściowego. Dane szeregowe mogą być wprowadzane i wyprowadzane za pomocą specjalnego układu we-wy.

Układ LM32900 wykonywany w technologii CMOS jest synchronizowany zegarem 20 MHz, przy czym cykl wewnętrzny jest równy 100 ns. Asembler i symulator dla LM32900 napisano w języku C i są one dostępne na mikrokomputerze IBM PC.



Rys. 5. Schemat blokowy układu mikroprocesora sygnałowego LM32900

Przykładem innego rozwiązania architektury mikroprocesora sygnałowego jest układ ADSP-2100 [7]. Ma on mniejszą liczbę szyn zewnętrznych (tylko jeden blok pamięci danych) oraz możliwość działania na liczbach zmiennoprzecinkowych.

Układ ADSP-2100 jest wyposażony w pięć szyn, przy czym cztery służą do współpracy z pamięciami zewnętrznymi, a piąta jest przeznaczona do przesłań wewnętrznych.

Mikroprocesor zawiera trzy niezależne bloki przetwarzania informacji: *ALU*, układ mnożący z akumulatorem i układ szybkiego przesuwania. Jednostka arytmetyczno-logiczna *ALU* umożliwia wykonanie operacji dodawania, odejmowania, dzielenia oraz funkcji logicznych. Wynik operacji jest zapamiętywany w rejestrze sprzęgającym *ALUFR* lub może być wykorzystany jako argument następnej operacji *ALU*.

Układ mnożący umożliwia wykonanie mnożenia, mnożenia z zapamiętaniem wyniku oraz mnożenia z odejmowaniem. Argumentami mnożenia są słowa 16-bitowe reprezentujące liczby w kodzie *NKB* lub *U*2 lub *NKB* i *U*2. Wynik mnożenia (32-bitowy) może być odjęty lub dodany od (do) zawartości 40-bitowego rejestru wyjściowego bloku mnożenia.

Układ przyspieszonego przesuwania umożliwia szybkie skalowanie ułamka liczby zmiennoprzecinkowej. Układ ten jest zrealizowany w postaci tablicy przyjmującej 16-bitowy argument, a dającej wynik 32-bitowy.

Argumenty operacji są pobierane z pamięci danych lub z pamięci danych i pamięci programu. Adresy tych argumentów są wytwarzane przez bloki funkcjonalne generatorów adresów danych, zawierające po 4 rejestry indeksowe, 4 rejestry modyfikacji adresu oraz 4 rejestry długości do przechowania rozmiaru adresowanej struktury danych. Działaniem mikroprocesora steruje program, którego instrukcje są wykonywane w jednym cyklu.

Oprogramowanie skrośne układu ADSP-2100 złożone z asemblera, programu łączącego, symulatora i systemu nadrzędnego jest m.in. dostępne na mikrokomputerze IBM PC. Dostępny jest również emulator układowy procesora.

Przedstawione układy LM32900 i ADSP-2100 wymagają dodatkowych bloków zewnętrznych, takich jak pamięć danych i programu. Bloki te zawiera jednoukładowy procesor sygnałowy μPD77230 [5], zrealizowany w technologii CMOS i umożliwiający wykonanie operacji zmiennoprzecinkowych dla argumentów 32-bitowych. Układ ten zawiera m.in.: stałoprzecinkowy układ mnożący 32 × 32 bity, blok obliczania wykładnika z układem skalowania ułamka, *ALU*, osiem 55-bitowych rejestrów roboczych, pamięć stałą programu o pojemności 2K × 32 bity, pamięć stałą danych 1K × 32 bity, dwie pamięci odczytywalno-zapisywalne dla danych o organizacji 512 × 32 bity każda, układ równoległego *we-wy* oraz układ szeregowego *we-wy*. Pamięć programu i danych może być rozszerzona za pomocą układów zewnętrznych.

Układ μPD77230 może pracować w systemach z wieloma procesorami sygnałowymi. Cykl podstawowy zegara wynosi 150 ns.

* * *

Do realizacji algorytmów wymagających częstego wykonywania operacji mnożenia danych kilkunastobitowych w czasie setek nanosekund jest konieczne stosowanie specjalnych układów mnożących. Układy te w postaci scalonych tablic mnożących wymagają zwykle odpowiedniej obudowy układami sterującymi. Układy takie jak MPY0-16K lub Am29323 są używane przeważnie jako bloki funkcjonalne będące częścią większej całości, np. procesora sygnałowego. Argumenty do mnożenia są więc z reguły dostarczane do nich z pamięci danych za pośrednictwem mikroprocesorów typu TDC1022 lub nie omówionego w niniejszym artykule układu Am29501 [2].

Oddzielny układ mnożący jest niepotrzebny, jeśli do projektu procesora wykorzystano układ Am29325. Jednak we wszystkich tych wypadkach sterowanie działaniem procesora odbywa się na poziomie mikroprogramu [4] i do jego budowy trzeba użyć wielu dodatkowych układów, m. in. sekwensera np. Am29331 [1]. Mniejszej liczby układów wymaga realizacja procesora sygnałowego przy użyciu układów LM32900, ADSP-2100 i μPD77230. Szybkość wykonywania obliczeń jest dla tych układów porównywalna, np. wykonanie 1024-punktowej szybkiej transformacji Fouriera wymaga w przybliżeniu 10 ms, przy czym najszybszy jest układ ADSP-2100 (7,2 ms). Do realizacji średnio skomplikowanych algorytmów może wystarczyć zastosowanie jednoukładowego procesora sygnałowego μPD77230. Cena wymienionych układów procesorów sygnałowych w 1986 roku wynosiła około 100 dolarów, natomiast układu Am29325 około 600 dolarów.

LITERATURA

[1] Am29300 Family Handbook. High Performance 32-Bit Building Block. AMD. April 1985
[2] Bipolar Microprocessor Logic and Interface. Am2900 Family 1983 Data Book. AMD. 1983
[3] Bipolar Microprocessor Logic and Interface. Am2900 Family 1985 Data Book. AMD 1985
[4] Budkowski S., Pawłowski M.: Problemy projektowania mikroprogramowanych urządzeń cyfrowych z użyciem mikroprocesorów segmentowych. Wydawnictwa Politechniki Warszawskiej. 1987
[5] Eichen B., Davis M.H., Kulp B.D.: Floating-point math integrated on chip makes DSP IC a standout. Electronic Design. 20 February 1986
[6] Electronics Design. No. 12. 1985
[7] LSI Data Book. TRW Inc., 1984
[8] Roesgen J., Tung S.: Moving memory off chip. DSP μP squeezes in more computational power. Electronic Design. 20 February 1986
[9] Schwartz M., Schiappacasse J., Baskerville G.: Signal processor's multiple memory buses shuttle data swiftly. Electronics Design. 20 February 1986.

Szanowna Redakcjo!

Opublikowany w numerach 10 i 11-12/88 *Informatyki* artykuł: W. Cellary, J. Kręglewski, R. Maćkowiak „Rodzina systemów mikrokomputerowych ELWRO-800" został napisany w 1985 roku i przedstawia stan prac nad mikrokomputerem ELWRO-800 ze stycznia 1986 roku. Po trzech latach, jakie upłynęły od tamtego momentu, pragniemy poinformować Czytelników o aktualnym stanie prac nad tym mikrokomputerem.

Dane dotyczące architektury i podstawowych parametrów systemu, zawarte w artykule, nie uległy zmianie. Obecnie są produkowane moduły: M-M16, M-M08, M-RAM, M-ECC, M-FLO, M-WDC, M-CRT, M-V24 i M-UKZ z przeznaczeniem dla ELWRO-800 stosowanym w systemach automatyki przemysłowej. Produkcję podjął Zakład Doświadczalny Instytutu Komputerowych Systemów Automatyki i Pomiarów we Wrocławiu.

Na zakończenie, w imieniu konstruktorów systemu ELWRO-800, możemy jedynie wyrazić żal, że system ten w odpowiednim czasie nie znalazł się w masowej produkcji.

WOJCIECH CELLARY
JERZY KRĘGLEWSKI

KAZIMIERZ KOZIARSKI
Instytut Automatyki
Systemów Energetycznych
Wrocław

Technologia konwersji oprogramowania cobolowego na IBM PC

Systemy przetwarzania danych eksploatowane na komputerach Odra i Riad w przeważającej większości zaprogramowano w Cobolu. Wraz z pojawieniem się w kraju mikrokomputerów PC wyposażonych w bogate oprogramowanie narzędziowe, zainteresowanie Cobolem wyraźnie zanikło. Doświadczenia autora wskazują jednak, że Cobol na mikrokomputerach PC w dalszym ciągu pozostaje bardzo dobrym narzędziem programowym i doskonale współpracuje z bazami danych.

W pierwszej części artykułu przedstawiono opis Cobolu na komputery PC ze szczególnym uwzględnieniem różnic z dialektami na Odrę i Riada. Druga część artykułu zawiera opis metody i techniki przenoszenia oprogramowania z Odry i Riada na PC. W zakończeniu omówiono równoległą eksploatację systemów przetwarzania danych w zestawie Odra-PC.

COBOL DLA KOMPUTERÓW PC

Cobol (ang. *Common Business Oriented Language*) jest językiem programowania szeroko stosowanym od ponad dwudziestu lat. Do dziś obowiązuje norma ANS X.23/1974 (American National Standard). Dzięki temu, że tworzenie kolejnych wersji języka przebiega zgodnie z normą, mimo ciągłego rozwoju jest to w zasadniczej warstwie ten sam język. Cobol ma składnię zbliżoną do języka angielskiego i jest językiem przeznaczonym do przetwarzania danych. Brak w nim funkcji matematycznych i arytmetyki zmiennoprzecinkowej. Ma natomiast mocno rozbudowaną obsługę masowych danych alokowanych na urządzeniach zewnętrznych. Na komputerach PC występuje kilka wersji języka. Obecnie w kraju najłatwiej dostępne są kompilatory:
MS-Cobol Compiler, wersja 1.0 (1982) i 1.1 (1983)
RM-Cobol Compiler wersja 2.1 (1985)

Struktura programu

Program w języku Cobol dzieli się na cztery obligatoryjne działy:
IDENTIFICATION DIVISION – identyfikacja programu,
ENVIRONMENT DIVISION – opis zestawu maszynowego,
DATA DIVISION – opis danych, które mają być przetwarzane,
PROCEDURE DIVISION – opis procedur wykonywanych w programie.
Technika pisania programów i podstawowe elementy języka (zbiór znaków, składnia, interpunkcja) są jednakowe dla wszystkich wersji Cobolu. Zmienia się tylko zakres słów kluczowych. Programy należy pisać w wierszach 80-kolumnowych. Na komputerach PC kolumny 1–6 oraz kolumny 73–80 nie są wykorzystywane i rozpoznawane przez kompilator.

W dziale identyfikacji *(IDENTIFICATION DIVISION)* jedyną obowiązującą klauzulą jest nazwa programu. Przy przejściu z Odry na PC należy z nazwy programu usunąć dwa ostatnie znaki określające priorytet wykonania. Pozostałe klauzule można tylko uaktualnić.

Dział zestawu maszynowego *(ENVIRONMENT DIVISION)* zawiera dwie sekcje *CONFIGURATION SECTION* i *INPUT-OUTPUT SECTION*. Na komputerach PC, *CONFIGURATION SECTION* nie jest obowiązkowa. W porównaniu z Odrą i Riadem większość klauzul przybiera deklaratywny charakter. Zmiany dotyczą przede wszystkim paragrafu *SPECIAL-NAMES*. PC nie obsługuje takich urządzeń, jak czytnik kart, czytnik i perforator taśmy papierowej, konsola operatorska. Odmienna jest również obsługa daty i czasu systemowego oraz słowa przełącznikowego (na Riadzie nie występuje). W paragrafie *I-O-CONTROL* dla wszystkich wersji jednakowe jest buforowanie pamięci. Na PC jest to jedyna klauzula w tym paragrafie. Nie ma możliwości kontroli pracy drukarki (braku papieru, przepełnienia strony). Natomiast kontrola współpracy z plikami dyskowymi jest przeniesiona do paragrafu *FILE-CONTROL*. Paragraf *FILE-CONTROL* wprowadza nazwy dla każdego pliku, identyfikuje nośnik i przydziela dla pliku odpowiednie urządzenie zewnętrzne. PC dopuszcza dwa rodzaje nośników: dyski *(DISK)* i drukarkę *(PRINTER)*. Pliki alokowane na innych urządzeniach (taśmy magnetyczne, czytnik kart, czytnik i perforator taśmy papierowej) w komputerach PC należy alokować jako *DISK*. Pliki mogą być oczywiście organizowane jako sekwencyjne, indeksowe, relatywne (losowe). Ponadto, na PC występuje klauzula *FILE STATUS* pozwalająca na kontrolę współpracy programu z plikami dyskowymi.

W dziale danych *(DATA DIVISION)* mogą występować następujące sekcje:
FILE SECTION (opis plików) – Odra, Riad, PC,
WORKING-STORAGE SECTION (opis pamięci roboczej) – Odra, Riad, PC
LINKAGE SECTION (sekcja łączników) – Odra, Riad, PC,
REPORT SECTION (opis raportów) – Riad,
SCREEN SECTION (opis formatów plansz) – PC.
Zgodnie z normą moduł raportów nie jest obowiązkowy i w obecnych wersjach Cobolu dla PC nie występuje. Dostępny jest jedynie na Riadzie. W programach wykorzystujących raportowanie, opisy plików należy przenieść do *FILE SECTION*, a sterowanie wydrukiem realizować w dziale procedur.

FILE SECTION zawiera definicje plików wykorzystywanych przez program. Każda definicja pliku składa się z opisu plików i występujących w nim rekordów. Plik jest opisywany klauzulą *FD* (*SD* odnoszona do plików sortowań nie występuje). Na tym poziomie obowiązkowe są zwroty *LABEL* oraz *VALUE OF FILE-ID*. Zwroty *BLOCK CONTAINS* oraz *DATA RECORD* są opcjonalne. Jeżeli wystąpią, to kompilator sprawdza ich poprawność. Pozostałe zwroty programów z Odry i Riada należy usunąć. Numer generacji pliku z programów dla Odry można umieścić w rozszerzeniu nazwy pliku. Dla plików dyskowych *LABEL RECORD* musi być *STANDARD*. Natomiast dla plików drukarki obowiązkowe jest umieszczenie zwrotu *LABEL RECORD OMITTED*. Ponadto, dla plików wydrukowych występuje opcjonalna klauzula *LINAGE* ułatwiająca rozmieszczenie wydruku na stronie tabulogramu. Struktury opisów rekordów i klauzule z nimi związane są prawie identyczne dla wszystkich wersji Cobolu. Na PC brak jedynie *OCCURS...DEPENDING*. W odniesieniu do Odry występuje różnica opisu drukarki. Dla PC i Riada błędne jest umieszczanie kombinacji znaków „V" jak również używanie „." (kropki) jako znaku wstawianego. Znak „." oznacza jednocześnie miejsce dziesiętne w polu. Dlatego też dla programów z Odry znak „V" występujący przy znaku „." należy usunąć. Zarówno dla PC jak i Riada pierwszy znak rekordu (zerowy) umożliwia sterowanie pracą drukarki.

Sekcje *LINKAGE* i *WORKING-STORAGE* pozostają bez zmian. Przy programach z Odry należy opisy poziomu 77 przenieść na początek sekcji *WORKING-STORAGE*. Kompilatory z Riada i Odry są wyczulone na kolejność opisu w pamięci roboczej.

SCREEN SECTION występuje tylko na komputerach PC. Znajduje zastosowanie przy opisie danych wyprowadzanych na ekran lub wprowadzanych z klawiatury. Całą komunikację z otoczeniem progra-

mów z Odry i Riada, realizowaną przez konsolę operatorską, czytnik kart, czytnik taśmy papierowej, należy umieścić w *SCREEN SECTION*. Nie jest zalecane realizowanie zasilania systemów za pomocą *SCREEN SECTION*. Jest to najsłabszy punkt Cobolu w wersji PC. Formatowanie plansz i danych jest mało atrakcyjne. Stosując inne oprogramowanie narzędziowe (np. dBase III Plus) zagadnienie to można rozwiązać o wiele ciekawiej.

W dziale procedur (*PROCEDURE DIVISION*) realizowane są wszystkie algorytmy przetwarzania programu, podzielone umownie przez programistę na sekcje i paragrafy. Dział procedur stanowi zasadnicze jądro zajmujące 60–80% objętości programu. Zakres zmian wprowadzanych w tym dziale jest najmniejszy i programy z Odry i Riada przeważnie przechodzą na PC bez poprawek. Instrukcje arytmetyczne są identyczne. Nie można jedynie używać zwrotu *CORRESPONDING* i należy działać na pojedynczych polach. Operatory logiczne i arytmetyczne są również identyczne. Dotyczy to też instrukcji logicznych *IF, GO, PERFORM*. Instrukcję działania na ciągach znakowych *EXAMINE* z Odry i Riada zastępuje *INSPECT* o podobnej budowie. Dodatkowo, na PC występują instrukcje *STRING* i *UNSTRING* przeznaczone do przegrupowania pól. Otwieranie i zamykanie plików jest zgodne, z wyjątkiem opcji specyficznych dla taśm magnetycznych. Instrukcje zapisu i odczytu pokrywają się w zakresie plików sekwencyjnych i wydrukowych. Nieco odmienna realizacja występuje przy plikach indeksowych i relatywnych.

Organizacja plików dyskowych

Pliki dyskowe Odry, Riada i PC mogą być organizowane jako sekwencyjne, relatywne (losowe) i indeksowe. Na komputerach PC pliki sekwencyjne dzielą się dodatkowo na sekwencyjne zwykłe i sekwencyjne wierszowe. Najpowszechniej stosowane pliki sekwencyjne zwykłe są zgodne dla wszystkich trzech komputerów. Zwykły plik sekwencyjny ma rekordy zaczynające się dwubajtowym słowem licznikowym, po którym następuje zawartość rekordu. Pliki sekwencyjne wierszowe występują tylko na komputerach PC, ich rekordy nie mają słowa licznikowego i są oddzielone od siebie znakiem nowego wiersza (*CR/LF*). Dla komputerów PC jest to plik klasy *SDF* (ang. *System Data Format*) tworzony przez zwykłe edytory tekstowe i wykorzystywany w wielu programach. Plik klasy *SDF* umożliwia przenoszenie danych systemu cobolowego do systemu dBase III Plus, Lotus, Basic lub innego oprogramowania narzędziowego. Pliki sekwencyjne zwykłe i wierszowe mają opcję dopisywania rekordów na końcu pliku.

Pliki indeksowe na PC mają odmienną organizację niż na Odrze i Riadzie. Dla każdego pliku indeksowego zadeklarowanego w programie generowane są dwa pliki dyskowe. Klauzula *VALUE OF FILE-ID* określa nazwę pliku danych. Nazwa ta z rozszerzeniem *KEY* tworzy nazwę pliku kluczowego. Każdy rekord pliku danych jest poprzedzony dwubajtowym polem licznikowym i jednobajtowym polem oznacznika kasowania rekordu. Do pliku danych przypisany jest plik kluczowy, przez który steruje się dostępem do danych. Dostęp do pliku o organizacji indeksowej może być sekwencyjny, przypadkowy i dynamiczny. Przy dostępie sekwencyjnym rekordy z pliku danych są dostępne według rosnących wartości klucza. Przypadkowy dostęp do pliku jest kontrolowany przez kolejne określenie wymaganej wartości klucza indeksowego. W wypadku dostępu dynamicznego można zmieniać dostęp z sekwencyjnego na przypadkowy i odwrotnie. Instrukcje obsługi pliku indeksowego (*READ, WRITE, DELETE, REWRITE* i *START*) są zgodne dla komputerów PC i Riad. Instrukcję *SEEK*, występującą dla Odry, można zastąpić instrukcją *START*.

Plik relatywny jest zbudowany z rekordów danych o stałej długości, równej długości największego rekordu zdefiniowanego dla tego pliku. Rekordy są rozróżniane przez numery w zakresie od 1 do 32767. Relatywne numery rekordów są domyślne i nie zawierają się w rekordach. Dostęp do pliku może być sekwencyjny, przypadkowy lub dynamiczny. W dostępie sekwencyjnym rekordy są przetwarzane w kolejności rosnących numerów rekordu. W dostępie przypadkowym kolejność jest kontrolowana przez określanie klucza relatywnego. W wypadku dostępu dynamicznego można dowolnie zmieniać dostęp przypadkowy i sekwencyjny. Przy tworzeniu plików relatywnych należy pamiętać o niepowtarzalności klucza relatywnego. Zgodność programowa Odry, Riada i PC jest taka sama jak w wypadku plików sekwencyjnych.

Wersja 1.00 Cobolu dla PC nie zawiera modułu sortowania i scalania plików. Sortowanie jest jednak możliwe w następnych wersjach kompilatora. Autonomicznych programów sortowania dużych plików na komputerach PC jest niewiele. Przy braku takiego programu sortowanie można z powodzeniem zastąpić indeksowaniem pliku. Wadą tej metody jest zajmowanie dodatkowego obszaru na dysku, ale jest ona szybsza. W programach z Odry i Riada wykonujących sortowanie wewnątrzprogramowe należy usunąć opis pliku na poziomie *SD*. Ponieważ indeksowanie odbywa się według jednego pola, może zachodzić potrzeba przegrupowania klucza sortowania w pliku wynikowym. Pole indeksowe (kluczowe) musi mieć wartość niepowtarzalną. Z tych względów bezpiecznie jest powiększyć klucz indeksowy o pole licznikowe zwiększane o 1 przy kolejnych instrukcjach zapisu rekordów w pliku indeksowym.

WSKAZÓWKI PRAKTYCZNE

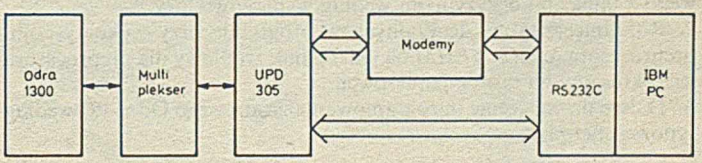
Poniżej omówiono techniczne szczegóły postępowania przy przenoszeniu oprogramowania w Cobolu na komputery PC i współpracę tych komputerów przy użytkowaniu systemów napisanych w Cobolu.

Kompilowanie i uruchamianie programów

Kompilowanie programów obejmuje dwie odrębne fazy (kompilacja i konsolidacja) nie wymagające specjalnych parametrów i dłuższego komentarza. W wyniku działania konsolidatora *LINK* otrzymuje się program ładowalny zapisany na pliku z rozszerzeniem *COM*. Programy Cobolu w trakcie działania korzystają z nakładki systemowej *COBRUN.EXE*. Kompilatory dla Odry i Riada potrafią wykryć wiele błędów składniowych, natomiast kompilator dla PC tego nie czyni – dlatego prosty błąd w rozbudowanym zdaniu jest sygnalizowany wieloma komunikatami. Prowadzi to czasami do zmniejszenia czytelności listy błędów. Zalecane jest używanie kompilatora o numerze wersji większej od 1.00, ponieważ programy przetłumaczone przez kompilator w wersji 1.00 nie działają prawidłowo na komputerach PC z pamięcią 640 KB (trzeba wtedy ograniczyć pamięć przez zainstalowanie dysku elektronicznego). Przy uruchamianiu programów można wykorzystywać instrukcję *TRACE*, która umożliwia śledzenie działania programu wskazując kolejność wykonywanych paragrafów. Powyższy opis dotyczy kompilatorów firmy Microsoft; działanie kompilatora RM-Cobol jest zdecydowanie odmienne.

Opis połączenia Odra-PC

Komputer Odra z komputerem PC można połączyć na dwa sposoby: zdalnie z wykorzystaniem kanału multipleksera i transmisji szeregowej oraz lokalnie przez kanały znakowe i transmisję równoległą.

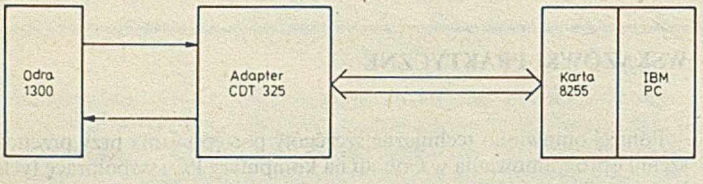


Rys. 1. Połączenie Odra-PC przez kanał multipleksera

Połączenie multiplekserowe [4, 6] (rys. 1) może być zrealizowane pod kontrolą *EGZEKUTORA* lub systemów operacyjnych GEORGE-2 i GEORGE-3. W wypadku programu nadzorczego *EGZEKUTOR* lub systemu operacyjnego GEORGE-2, Odra wymaga dodatkowo programu *ON-LINE*. Dla Odry pracującej pod nadzorem systemu GEORGE-3 przesyłanie plików jest realizowane za pomocą poleceń *LISTFILE* i *INPUT*. Przesyłanie danych może przebiegać z szybkością 300–9600 bodów. Do uzyskania szybkości powyżej 1200 bodów wymagana jest modernizacja adaptera linii UPD-305. Podobny typ połączenia wykonuje się dla Riada [10].

Połączenie lokalne [8] przedstawiono na rys. 2. Komputer PC jest dołączony przez kanały czytnika i perforatora taśmy papierowej za

pomocą adaptera transmisji równoległej. Odra traktuje PC jako emulator CDT-325 do czytania i perforowania dziesiętnego i binarnego. Operacja perforowania odpowiada zapisywaniu pliku na dysk, a operacja czytania taśmy papierowej odpowiada odczytywaniu plików z dysku. Połączenie działa przy równoległej pracy programów na obu komputerach Odra i PC. Po stronie PC jest to program nadawczo-odbiorczy do portu 8255, natomiast po stronie Odry jest to dowolny program (systemowy lub użytkowy) wykonujący procedury czytania i perforowania taśmy papierowej. Przesyłanie przebiega z szybkością 30 000–50 000 bodów. Szybkość transmisji jest uzależniona od modelu PC i wyboru nośnika (dysk stały, dyskietka). Ze względu na większą szybkość transmisji i lepszą niezawodność, połączenie lokalne Odra-PC jest dla systemów przetwarzania danych korzystniejsze.



Rys. 2. Połączenie Odra-PC w kanałach czytnika i perforatora CDT-325

Emulacja oprogramowania Odra-PC

Przeniesienie oprogramowania źródłowego przez połączenie lokalne Odra-PC nie wymaga specjalnych zabiegów. Programy źródłowe z Odry na kartach perforowanych, taśmie magnetycznej lub dyskach wymiennych należy wyprowadzić na perforator taśmy papierowej. Można to wykonać trzema sposobami:

- programem *XRCP* – wyprowadzanie danych zawartych na kartach perforowanych na taśmę papierową,
- kompilatorem *XEKB* – kopiowanie postaci źródłowej z jednego nośnika na drugi,
- preprocesorem *XEHE* – określenie taśmy papierowej jako pliku wynikowego.

W systemie Odra występuje preprocesor tablic decyzyjnych *XEHE/XEHM* umożliwiający pisanie programów z wykorzystaniem techniki tablic decyzyjnych. Jest to bardzo wygodne, niezawodne i często stosowane narzędzie programowe. Dla programów z tablicami decyzyjnymi proces emulacji przebiega nieco odmiennie. Do komputera PC nadawana jest postać źródłowa programu po przetworzeniu przez procesor *XEHE/XEHM* – z rozwiniętymi tablicami decyzyjnymi na sekcje działu procedur. Rozwinięcie tablic decyzyjnych musi być generowane metodą skoków, w przeciwnym wypadku preprocesor generuje odwołanie do podprogramów, które nie występują na PC. Wykorzystując połączenie lokalne w odwrotnym kierunku można na komputerze PC pisać programy za pomocą tablic decyzyjnych. Jest to spore ułatwienie w pracy programisty i może być realizowane w kolejnych krokach:

1. Napisanie na PC dowolnym edytorem tekstowym programu źródłowego z tablicami decyzyjnymi według wymogów Odry.
2. Rozwinięcie tablic decyzyjnych w programie przy wykorzystaniu preprocesora *XEHE/XEHM* na Odrze; plik źródłowy dla preprocesora jest określany na taśmie papierowej.
3. Dalsze przenoszenie oprogramowania źródłowego Odra-PC według typowej metody.

Programy źródłowe z Odry przeniesione na PC można kompilować bezpośrednio kompilatorem Cobolu dla PC. Na podstawie listy błędów kompilacji można sprawnie dokonać zmian w programie. Zakres zmian jest zagadnieniem indywidualnym dla każdego programu i zależy od wielu różnorodnych czynników. Największe znaczenie ma przy tym technika pisania programu, wykorzystanie urządzeń zewnętrznych i organizacja plików. W programie drukowania kartoteki materiałowej i liczącym dla Odry 410 wierszy źródłowych dokonano następujących zmian: usunięto 23 wiersze, dodano 57 nowych wierszy, a 18 wierszy zmodyfikowano. Nawet w programach liczących powyżej 1500 wierszy, wprowadzenie zmian sprowadza się do kilku godzin pracy. Łącznie czas potrzebny na przełożenie systemu liczącego 30–40 programów zamyka się w granicach trzech miesięcy pracy jednej osoby, co obejmuje prace związane z wprowadzeniem zmian w programach i ich częściową modernizacją. Nie występują natomiast prawie wcale pracochłonne czynności w zakresie testowania i uruchamiania systemu.

Współdziałanie Cobolu i dBase III Plus

Na komputerach Odra i Riad zasilanie systemów przetwarzania w dane jest oparte na nośniku magnetycznym zapisywanym na Merze 9150 lub – na kartach perforowanych. Wprowadzenie danych do komputera PC musi być zorganizowane inaczej. Cobol nie zawiera modułu wprowadzania danych i nie jest przystosowany do realizacji tego zadania. Do wykonania tej funkcji można zastosować pakiet dBase III Plus. W dBase III Plus łatwo tworzy się formularze ekranowe i organizuje obsługę programową klawiatury. Pliki pakietu dBase III Plus nie nadają się jednak do bezpośredniego wykorzystania przez programy Cobolu. Plik bazy danych dBase zawiera na początku opis struktury danych, a dalej rekordy danych według układu sekwencyjnego. Polecenie *COPY...FILE...SDF* w systemie dBase umożliwia przekazanie danych z pliku *DBF* do pliku *TXT*, stanowiącego odpowiednik pliku sekwencyjnego wierszowego w języku Cobol. Dane zawarte w pliku sekwencyjnym wierszowym utworzonym w dBase III Plus muszą być przejmowane w programie Cobolu pośrednio z wykorzystaniem pamięci roboczej *WORKING-STORAGE*. Wynika to z odmiennego zapisu danych w dBase i w Cobolu.

System dBase III Plus zapisuje dane rozwlekle i odmiennie niż czynią to języki programowania. W polach numerycznych, oznaczenie miejsca dziesiętnego „." (kropką) jest zapisywane do bazy, co niepotrzebnie zwiększa jej rozmiar. W polach dziesiętnych, na miejscu zer nieznaczących znajdują się spacje. Dlatego z pól numerycznych dBase III Plus należy usunąć znak „." przez przegrupowanie pól i instrukcją Cobolu *INSPECT* zamienić spacje na nieznaczących miejscach pola dziesiętnego na nieznaczące zera. Z pakietu dBase III Plus można również wykorzystać polecenie sortowania pliku i przekazywać dane uporządkowane.

Eksploatacja równoległa Odra-PC

Stosując moduł połączenia lokalnego można prowadzić eksploatację systemów w układzie rozproszonym. W systemie Odra wystarczy wykonać proste programy w Cobolu obsługujące konwersję plików między nośnikami magnetycznymi a czytnikiem i perforatorem taśmy papierowej. Ponadto, programy na Odrze zasilające system przetwarzania w dane należy rozszerzyć o wejścia z taśmy papierowej. Uzyskuje się w ten sposób pełne sprzężenie systemów Odra i PC. Dane przygotowane na PC można wprowadzać do systemu Odra lub systemu PC. Dane przetworzone na PC można przesyłać do Odry w celu obsługi dużych tabulogramów. Drukowanie na PC jest 10–15-krotnie wolniejsze, co przy dużych tabulogramach stanowi spore utrudnienie w pracy.

* * *

Na komputerach Odra i Riad pracuje sporo dobrych i sprawdzonych systemów użytkowych. Wykorzystując przedstawioną metodę można znacznie przyspieszyć tworzenie oprogramowania aplikacyjnego na komputery PC, przy jednoczesnym obniżeniu kosztów. W wielu wypadkach niepotrzebnie przygotowuje się na te komputery systemy aplikacyjne od podstaw, zapominając o tym, że na komputerach Odra i Riad pracuje od lat podobne oprogramowanie.

Cobol dobrze znosi próbę czasu. Do sekwencyjnej obsługi masowych zbiorów danych brak, jak dotąd, lepszego języka programowania. Kolejne wersje Cobolu na PC zmierzają w kierunku utworzenia oprogramowania przyjaznego, co w niedalekiej przyszłości może doprowadzić do ponownego zainteresowania Cobolem w kraju.

LITERATURA

[1] Cobol na mikrokomputery – wersja MS-Cobol. Centrum Usług Informatycznych, Technicznych i Ekonomiczno-Organizacyjnych, Gdańsk Oliwa, 1987
[2] Cobol – opis języka. Instytut Komputerowych Systemów Automatyki i Pomiarów, Wrocław, 1987
[3] Cobol – programy pomocnicze. Instytut Komputerowych Systemów Automatyki i Pomiarów, Wrocław, 1979
[4] CSK: Materiały ofertowe. Gdynia, 1986
[5] Koziarski K., Kurczalski P.: Emulacja oprogramowania z mc Odra 1300 na mc Riad OS/MVT. Instytut Automatyki Systemów Energetycznych, Oprac. nr 64921, Wrocław, 1984
[6] O.K.: Materiały ofertowe. Warszawa, 1986
[7] OS/JS Cobol – opis języka. Mera-Elwro, Wrocław, 1987
[8] Próchniak T.: Dokumentacja techniczno-ruchowa adaptera lokalnej współpracy mikrokomputera z systemem Odra 1300. Instytut Automatyki Systemów Energetycznych. Oprac. nr 319, Wrocław, 1986
[9] Zastosowanie dBase III Plus. Intersoft, Warszawa, 1987
[10] ZETO-ZOWAR: Materiały ofertowe. Warszawa, 1986

WIESŁAW NOWIŃSKI
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

Algorytmy rekonstruowania obrazów za pomocą metod transformacyjnych

W poprzednim artykule [3] opisano problematykę z zakresu rekonstruowania obrazów oraz przedstawiono ogólny schemat systemu rekonstruowania wraz z opisem jego układów składowych. Celem niniejszego artykułu jest scharakteryzowanie transformacyjnych metod rekonstruowania obrazów oraz opartych na tych metodach algorytmów i sposobu ich implementacji.

PODSTAWY TRANSFORMACYJNYCH METOD REKONSTRUOWANIA

Transformacyjne metody rekonstruowania obrazów są oparte na analizie matematycznej, a w szczególności na teorii transformacji Fouriera. W metodach tych wykorzystuje się zależności wyrażające obraz f w funkcji wektora pomiarów y. Istniejące metody transformacyjne wykorzystują odwrotną transformatę Radona (transformata Radona R jest całką wzdłuż promienia ze współczynnika tłumienia, który jest rekonstruowany – wynik całkowania zwany jest sumą wzdłuż promienia) lub związki między transformatami Radona i Fouriera. W celu implementacji tych metod należy otrzymane zależności między f a y zdyskretyzować.

Niech R^{-1} będzie odwrotną transformatą Radona, tzn.

$$R^{-1} Rf = f.$$

Wobec tego, znajomość dokładnych wartości danych pomiarowych

$$p = Rf$$

umożliwia wyznaczenie obrazu z zależności:

$$f = R^{-1} p$$

Okazuje się, że operator R^{-1} można wyrazić jako złożenie czterech operatorów [2]:

$$R^{-1} = C B H D \qquad (1)$$

gdzie D jest pewnym operatorem różniczkowania cząstkowego, H – transformacją Hilberta, B – operatorem rzutowania zwrotnego (opisanym w dalszej części artykułu) oraz C operatorem normalizacyjnym (stałą). Zależność (1) określa z matematycznego punktu widzenia sposób dokładnego wyznaczenia funkcji obrazu, jednakże nie rozwiązuje problemu rekonstruowania, gdyż:
– w tomografii komputerowej dysponujemy skończonym wektorem pomiarów, natomiast zależność (1) wymaga znajomości wszystkich możliwych sum wzdłuż promienia;
– zależność (1) jest prawdziwa dla dokładnych danych pomiarowych, natomiast w tomografii komputerowej wektor pomiarowy jest obarczony błędem.

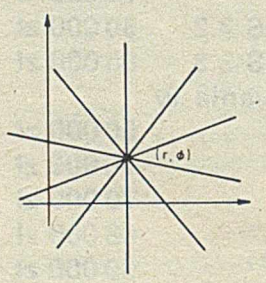
Zależność (1) jest formułą matematyczną, według której należy opracować odpowiedni algorytm. Aproksymacja odwrotnej transformaty Radona operatorem rzutowania zwrotnego nazywa się metodą rzutowania zwrotnego, natomiast aproksymacja w zależności (1) różniczkowania i transformacji Hilberta splotem nazywa się metodą splotu. Bezpośrednie zależności między transformatą Radona i Fouriera prowadzą do dwóch dalszych metod: Fouriera i filtrowanego rzutowania zwrotnego.

W całym artykule operatory i transformaty (np. R, B, F...) oznaczono czcionką o stopień większą od pozostałego składu (przyp. red.)

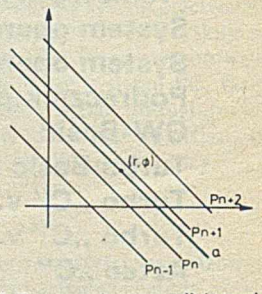
RZUTOWANIE ZWROTNE

Najprostszym algorytmem rekonstruowania jest przybliżenie gęstości danego punktu obrazu (r, φ) przez sumę sum wzdłuż promienia dla wszystkich prostych przechodzących przez ten punkt (rys. 1). Metoda ta jest zwana metodą rzutowania zwrotnego lub metodą sumacyjną.

Niech a będzie jedną z pęku prostych przechodzących przez punkt (r, φ). W celu wyznaczenia wartości sumy wzdłuż promienia odpowiadającej prostej a, należy z rzutu, którego promienie są równoległe do tej prostej, wybrać te dwa promienie, między którymi leży prosta a (rys. 2). Następnie, biorąc wartości sumy wzdłuż promienia p_n i p_{n+1}, dla tych promieni wyznaczamy wartość sumy wzdłuż promienia dla prostej a przez interpolację. Najczęściej wykorzystuje się interpolację liniową i interpolację metodą najbliższego sąsiedztwa, aczkolwiek stosuje się również wielomiany Lagrange'a i funkcje sklejane.



Rys. 1. Interpretacja rzutowania zwrotnego



Rys. 2. Wyznaczanie sumy wzdłuż promienia dla prostej a

Zauważmy, że działanie operacji rzutowania zwrotnego jest niezależne od położenia punktu (r, φ). Umożliwia to równoległą implementację algorytmu rekonstruowania obrazów metodą rzutowania zwrotnego za pomocą architektury typu SIMD.

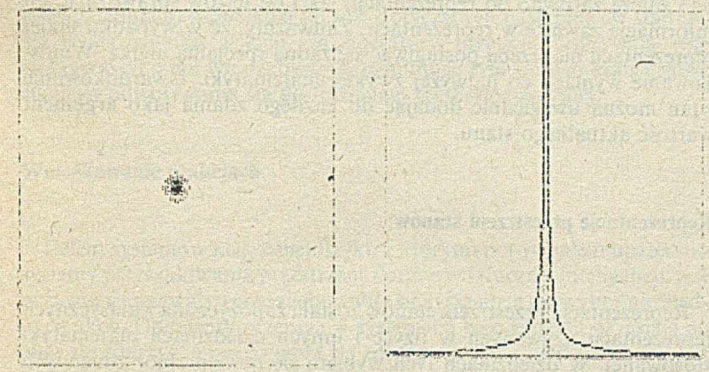
Przy implementacji metody rzutowania zwrotnego rozróżnia się dwa zasadnicze podejścia, określając je jako rzutowanie zwrotne sterowane rzutami i rzutowanie zwrotne sterowane pikselami (czyli punktami obrazu). Rzutowanie zwrotne sterowane rzutami polega na sekwencyjnym przetwarzaniu kolejnych rzutów (tzn. po pomiarze kolejnego rzutu przez układ pomiarowy pozostałe układy systemu rekonstruowania mogą ten rzut przetwarzać), natomiast w rzutowaniu zwrotnym sterowanym pikselami przetwarza się równoległe rzuty i oblicza gęstości dla wszystkich punktów obrazu przy znajomości całego wektora pomiarowego. Załóżmy, że z danych M rzutów rekonstruuje się obraz w J punktach (tzn. przyjmuje się, że obraz został zdyskretyzowany w J punktach obrazu i dla każdego punktu obrazu oblicza się wartość operacji rzutowania zwrotnego). Wówczas do zrekonstruowania obrazu przy stosowaniu rzutowania zwrotnego sterowanego rzutami potrzeba przynajmniej $(2M\text{-}1)J$ dostępów do pamięci: MJ dla zapisu (dla uaktualnienia w każdym rzucie wartości gęstości dla każdego punktu obrazu) i $(M\text{-}1)J$ dla odczytu (wartości gęstości dla każdego punktu obrazu przed kolejnym uaktualnieniem). Natomiast przy stosowaniu rzutowania zwrotnego sterowanego pikselami potrzeba jedynie J dostę-

pów do pamięci – do zapisania wartości gęstości dla każdego punktu obrazu. Wobec tego rzutowanie zwrotne sterowane pikselami jest potencjalnie $2M$-1 razy szybsze, niż rzutowanie zwrotne sterowane rzutami.

Niech δ będzie deltą Diraca umieszczoną w początku układu współrzędnych biegunowych (r, φ). Wówczas:

$$[B\delta](r, \varphi) = 1/r$$

czyli operacja rzutowania zwrotnego aproksymuje impuls i równocześnie powoduje jego rozmycie. Ilustruje to rys. 3 otrzymany na komputerze PC/AT z kartą CGA (tylko cztery poziomy szarości). Dla dowolnego obiektu operacja rzutowania zwrotnego powoduje rozmycie brzegów zrekonstruowanego obrazu obiektu i zmniejszenie kontrastu obrazu. Degradacja jakości obrazu wynika z faktu, iż f przybliżamy przez By, natomiast pomija się pozostałe operacje z zależności (1). Pomimo tej istotnej wady metoda rzutowania zwrotnego stanowi podstawę tomografii tradycyjnej, aczkolwiek w tomografii komputerowej należy stosować dokładniejsze metody, jak np. metodę splotu.



a) b)

Rys. 3. Zrekonstruowana delta Diraca: a) obraz, b) krzywa gęstości

METODA SPLOTU

Podstawowa trudność w implementacji wzoru (1) leży w obliczaniu transformaty Hilberta zawierającej całkę niewłaściwą. Dlatego aproksymuje się we wzorze (1) różniczkowanie i transformację Hilberta splotem znacznie prostszym numerycznie do obliczenia. Prowadzi to do następującej zależności:

$$\hat{f} = C\,B\,(y * q) \tag{2}$$

gdzie znak „$*$" oznacza splot, q zaś jest pewną funkcją splatającą. Opierając się na twierdzeniu o splocie można tak dobierać funkcję splatającą q, aby transformata splotu a tym samym przybliżenie zrekonstruowanego obrazu $\hat{f}$ spełniało określone warunki. Przykładowo, jeśli funkcja obrazu f jest ograniczona do pewnego pasma, to w pobliżu częstotliwości krańcowych tego pasma dane pomiarowe są obarczone największym szumem. Dobierając funkcję splatającą tak, aby jej transformata Fouriera w pobliżu częstotliwości krańcowych była bliska zera otrzymujemy tłumienie szumu tym większe, im wartości tej transformaty są bliższe zera.

W metodzie rzutowania zwrotnego rzutowany zwrotnie jest wektor pomiarowy y, natomiast w metodzie splotu należy rzutować zwrotnie wektor y spleciony z funkcją splatającą q. Wobec tego przy implementacji metody splotu można oprzeć się na implementacji metody rzutowania zwrotnego, zastępując wektor pomiarowy y przez splot $y*q$, który można efektywnie obliczać wykorzystując np. szybką transformację Fouriera.

METODA FOURIERA

Metoda Fouriera nie ma już tak prostej interpretacji geometrycznej, jak np. metoda rzutowania zwrotnego. Metoda ta oparta jest na twierdzeniu o projekcji określającym związek między transformatami Radona i Fouriera. Niech F będzie jednowymiarową, F_2 zaś dwuwymiarową transformatą Fouriera. Wówczas spełniony jest następujący związek [1]:

$$F_2 = F\,R \tag{3}$$

Uwzględniając zależności

$$F_2^{-1} F_2 f = f$$
$$p = Rf$$

ze związku (3) otrzymujemy odpowiednią regułę rekonstruowania, mianowicie

$$f = F_2^{-1} F p \tag{4}$$

Operacja Fp jest rozumiana jako transformata Fouriera względem pierwszej zmiennej (związanej z przesunięciem liniowym kolejnych promieni wiązki pomiarowej) funkcji p. W tomografii komputerowej dysponujemy jedynie skończonym wektorem pomiarowym y (dokładne dane pomiarowe p nie są znane), więc z równania (4) otrzymujemy przybliżenie $\hat{f}$ obrazu f.

Implementacja metody Fouriera sprowadza się do zastosowania szybkiej transformacji Fouriera: jednowymiarowej prostej i dwuwymiarowej odwrotnej.

METODA FILTROWANEGO RZUTOWANIA ZWROTNEGO

Metoda filtrowanego rzutowania zwrotnego, podobnie jak metoda Fouriera, jest oparta na pewnym związku między transformacjami Fouriera i Radona, mianowicie

$$S = F_2 f = |R|\, F_2\, B\, R f \tag{5}$$

gdzie R jest zmienną w dziedzinie transformaty Fouriera odpowiadającą zmiennej biegunowej r. Po obliczeniu wartości S, obraz wyznacza się z równania

$$f = F_2^{-1} S, \qquad \text{dla} \qquad R \neq 0$$

Również i w wypadku metody filtrowanego rzutowania zwrotnego można wyznaczyć jedynie przybliżenie $\hat{f}$ obrazu f, jako że dla równania (5) nie dysponujemy dokładnymi danymi pomiarowymi.

Implementacja metody sprowadza się do implementacji następujących operacji:
- rzutowania zwrotnego wektora pomiarowego,
- dwuwymiarowej prostej transformacji Fouriera (za pomocą szybkiego przekształcenia Fouriera),
- skalowania (mnożenia przez $R \neq 0$),
- dwuwymiarowej odwrotnej transformacji Fouriera (za pomocą szybkiego przekształcenia Fouriera).

* * *

W artykule scharakteryzowano transformacyjne metody rekonstruowania obrazów: rzutowania zwrotnego, splotu, Fouriera i filtrowanego rzutowania zwrotnego. Mniejsze koszty, liczone jako czas pracy i niezbędna pojemność pamięci komputera, ponoszone przy implementacji algorytmów opartych na metodach transformacyjnych powodują, że właśnie te algorytmy są powszechnie stosowane w dostępnych handlowo tomografach komputerowych. Ponadto, metody transformacyjne umożliwiają rekonstruowanie jedynie części obrazu (zauważmy, że operację rzutowania zwrotnego można wykonać dla każdej liczby dowolnie wybranych punktów obrazu). Wadą metod transformacyjnych jest konieczność dostarczenia dużej liczby dokładnych danych pomiarowych. Naraża to pacjenta na otrzymanie większej (niż w wypadku stosowania metod opartych na rozwinięciu w szereg) dawki szkodliwego promieniowania, ponadto wymaga właściwego wstępnego przetwarzania obrazów.

LITERATURA

[1] Ludwig D.: The Random transform on Euclidean Space. Commun. Pure Appl. Math., Vol. 19, 1966

[2] Nowiński W.: Metody rekonstrukcji obrazów w tomografii komputerowej. Prace IPI PAN, nr 597, Warszawa, 1986

[3] Nowiński W.: Rekonstruowanie obrazów w tomografii komputerowej. Informatyka, nr 3, 1989.

HERBERT A. SIMON
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, stan Pensylwania
USA

Procesy przeszukiwania i wnioskowania w rozwiązywaniu problemów (2)

Pierwsza część artykułu była poświęcona systemom wnioskowania. W drugiej – omówiono systemy przeszukiwania przestrzeni rozwiązań i porównano przydatność obu rodzajów systemów do rozwiązywania rzeczywistych problemów.

ROZWIĄZYWANIE PROBLEMÓW JAKO PRZESZUKIWANIE PRZESTRZENI ROZWIĄZAŃ

Rozwiązanie przykładowego problemu przedstawionego w poprzedniej części artykułu (zadanie mieszania cieczy) przebiega w najbardziej naturalny sposób jako proces wnioskowania. Zauważyliśmy jednak, że rozwiązanie tego problemu może być również traktowane jako proces przeszukiwania przestrzeni rozwiązań. Przestrzenią rozwiązań – po ustaleniu reprezentacji problemu i dodaniu praw zachowania – jest przestrzeń dopuszczalnych wartości zmiennych. Rozbieżność między aktualną sytuacją a pożądanym celem ulega zmniejszeniu za każdym razem, gdy zostanie znaleziona wartość jednej zmiennej. Jeśli wszystkie różnice zostaną wyeliminowane, to problem uznaje się za rozwiązany.

Rozważany problem kwalifikuje się jako zadanie odpowiednie dla metody zwanej analizą środków i celów (ang. *means-ends analysis*), ponieważ wyeliminowanie kolejnej różnicy (czyli rozwiązanie równania ze względu na jedną niewiadomą) nigdy nie doprowadzi do ponownego pojawienia się tych rozbieżności, które już zostały wyeliminowane. Nie ma także znaczenia kolejność eliminacji: jeśli tylko dysponujemy operatorem rozwiązującym równanie ze względu na jakąś zmienną (tzn. jeśli tylko zmienna ta jest jedyną niewiadomą w równaniu), to dany operator może być natychmiast zastosowany.

Bardziej typowy problem przeszukiwania przestrzeni rozwiązań

Pomimo tego, co już zostało powiedziane, zadanie mieszania cieczy jest raczej problemem nietypowym dla podejścia opartego na przeszukiwaniu przestrzeni rozwiązań. Do zilustrowania procesów przeszukiwania znacznie lepiej nadaje się „szacowny" problem misjonarzy i ludożerców. Problem ten jest zazwyczaj reprezentowany za pomocą zbioru stanów, z których każdy odpowiada konkretnemu rozmieszczeniu misjonarzy, ludożerców i łodzi na obu brzegach rzeki. Operatory przejścia działają na te reprezentacje stanów, modyfikując rozmieszczenia obiektów po obu stronach rzeki, tzn. przeprowadzają proces od jednego stanu do innego.

Prawdy mogą być określane za pomocą reprezentacji przestrzeni stanów jedynie warunkowo. Możemy na przykład powiedzieć warunkowo *„w stanie A, na lewym brzegu jest dwóch ludożerców"*, lecz nie możemy powiedzieć bezwarunkowo *„na lewym brzegu jest dwóch ludożerców"*. Takie uwarunkowanie asercji nie powoduje żadnych trudności, jeśli posługujemy się reprezentacją przestrzeni stanów. Na dowolnym etapie procesu rozwiązywania zostaje wygenerowany jakiś stan *S*. Ponieważ stan *S* zawiera – jawnie lub niejawnie – prawdę o samym sobie, możemy więc skonstruować takie procesy badania lub wnioskowania, które będą wyprowadzać wnioski prawdziwe dla stanu *S*.

Reprezentację można uważać za podstawę wszelkich ważnych zdań generowanych w trakcie rozwiązywania problemu. Zdanie *„na lewym brzegu jest dwóch ludożerców"* jest reprezentowane (w mniejszym lub większym stopniu) jawnie, natomiast zdanie *„na lewym brzegu jest więcej ludożerców niż misjonarzy"* jest reprezentowane (w mniejszym lub większym stopniu) niejawnie. Oznacza to, że proces badania ważności pierwszego zdania będzie prostszy i bardziej bezpośredni niż proces badania ważności drugiego. Określenie *„jawny"* i *„niejawny"* są względne i zależą zarówno od reprezentacji, jak i od testów sprawdzających informacje zawarte w reprezentacji. Zauważmy, że w wypadku takiej reprezentacji nie trzeba posługiwać się żadną specjalną logiką. Wnioskowanie wymaga co najwyżej zwykłej matematyki. Uwarunkowanie zdań można uwzględnić dodając do każdego zdania jako argument wartość aktualnego stanu.

Reprezentacje przestrzeni stanów

Reprezentacja przestrzeni stanów została zapożyczona z klasycznych reprezentacji używanych w fizyce i innych dziedzinach matematyki stosowanej. W dziedzinach tych wybiera się pewien zbiór zmiennych bazowych (położenie i prędkość w wypadku klasycznej dynamiki), a każdy punkt w czasoprzestrzeni opisuje się za pomocą wektora wartości tych zmiennych. Prawa dotyczące systemu, zapisane zwykle w postaci równań różniczkowych, są operatorami przejścia. Zwyczajne wnioskowanie matematyczne pozwala wyciągać wnioski o wartościach zmiennych różnych od zmiennych bazowych.

Najważniejszą cechą reprezentacji przestrzeni stanów jest oparcie się na stosunkowo małym i „ekonomicznym" zbiorze zmiennych bazowych – zapewnia to budowę względnie prostych operatorów przejścia między stanami. Aby wygenerować jakiś stan, nie trzeba wyprowadzać wszystkich zdań prawdziwych dla danego stanu ani usuwać zdań, które były prawdziwe w poprzednim stanie i które przestały być prawdziwe. Wystarczy skoncentrować się jedynie na wartościach zmiennych bazowych.

Oczywiście nie wymaga się, żeby reprezentacja była w powyższym sensie minimalna. W pewnych okolicznościach jest wygodnie uwzględnić również inne (redundantne) zmienne i uaktualniać ich wartości w tym samym czasie, gdy uaktualnia się wartości zmiennych bazowych. Warto podkreślić, że takie redundantne uaktualnianie może być wygodne, lecz z pewnością nie jest konieczne. Z tego faktu wynika duża przydatność klasycznej matematyki do opisu stosunkowo złożonych systemów.

Systemy podobne do systemu *STRIPS*

Ekonomiczność reprezentacji jest wykorzystywana efektywnie w systemach podobnych do systemu *STRIPS* [1]. Aby uniknąć uaktualniania każdego zdania, prawdziwego w jakimś konkretnym stanie, reprezentację ogranicza się zwykle do zbioru zmiennych bazowych, które są uaktualniane za pomocą operatorów przejścia, dodających lub usuwających elementy (zdania) z reprezentacji. Właściwości stanu, których baza nie opisuje, mogą być z bazy wywnioskowane na podstawie przesłanek odnoszących się do poprzedniego stanu. Nie jest przy tym potrzebna żadna niemonotoniczna logika, gdyż prawda jest zachowana wewnątrz każdego stanu. Jeśli zaistnieje potrzeba zapamiętania informacji o stanach różnych od stanu bieżącego, to informację tę można po prostu zaetykietować nazwą stanu, w którym ona obowiązuje.

Systemy podobne do systemu *STRIPS* często zapewniają jeszcze inny rodzaj wydajności obliczeniowej. W wielu systemach operatory przejścia, działając na określony stan, zmieniają tylko niektóre spośród jego właściwości. Właściwości stanu, które nie ulegają zmianie w następnym stanie, są zapamiętywane w reprezentacji i nie wymagają dalszego przetworzenia. Uważa się wówczas, że obowiązują one w następniku danego stanu, a nie w jego poprzedniku. Układy równań różniczkowych i różnicowych w fizyce i ekonomii rzadko mają tę przydatną cechę, ponieważ wartości wszystkich zmiennych stanu zmieniają się w każdej „chwili". W wielu problemach z dziedziny sztucznej inteligencji operatory przejścia oddziałują w danej chwili tylko na niektóre składniki stanu.

W literaturze poświęconej sztucznej inteligencji systemy podobne do systemu *STRIPS* były często traktowane jako systemy wnioskowania, a nie jako systemy przeszukiwania przestrzeni rozwiązań (oryginalna publikacja poświęcona systemowi *STRIPS* [1] nosi podtytuł „Nowe podejście do wykorzystania dowodzenia twierdzeń w rozwiązywaniu problemów"). Przedstawiona tutaj analiza sugeruje raczej, że siłą tego rodzaju systemów jest ich umiejętność modelowania procesu przechodzenia systemu przez przestrzeń stanów bez uaktualniania (za pomocą jawnej dedukcji) wszystkich faktów dotyczących następników stanów. Ta cecha wynika z tego, że operatory przejścia odgrywają rolę efektywnych reguł wnioskowania, eliminując tym samym konieczność ciągłego odwoływania się do formalnej logiki dedukcyjnej.

Wnioskowanie o akcjach

Celem systemów klasycznej fizyki, z którymi porównałem uprzednio systemy przeszukiwania przestrzeni stanów w sztucznej inteligencji, jest oczywiście przewidywanie zachowania się systemu, a nie wybór niezbędnych akcji. Operatory przejścia w systemach tego rodzaju określają co **musi** się zdarzyć, a nie co **może** się zdarzyć. Logika tych systemów jest deterministyczna, a nie – niedeterministyczna. Być może więc, gdy nie mamy do czynienia z zadaniami przewidywania, lecz z zadaniami planowania lub dokonywania wyboru, potrzebne są inne metody. Być może konieczność wprowadzenia logiki modalnej lub jakiejś innej logiki niemonotonicznej wynika z potrzeby uwzględnienia – oprócz świata faktycznego – również alternatywnych światów możliwych.

O tym, że jest inaczej, możemy przekonać się analizując dowolny system wybrany spośród licznych normatywnych systemów formalnych, które zostały stworzone w ekonomii i badaniach operacyjnych w celu wyznaczania strategii optymalizujących lub spełniających warunki dostateczności. Takie techniki, jak na przykład programowanie dynamiczne, posługują się reprezentacjami przestrzeni stanów z małymi zbiorami zmiennych bazowych i kwalifikują wszystkie asercje dotyczące wartości zmiennych według stanów, do których się one odnoszą. Ewidentnym potwierdzeniem tego faktu są indeksy czasowe, które pojawiają się w każdym równaniu. Stosowane w tym wypadku metody wnioskowania są po prostu metodami zwykłej matematyki; nie wymagają one żadnej specjalnej logiki. We współczesnej literaturze z dziedziny logiki podejście to pojawia się pod hasłem „semantyka światów możliwych" (ang. *possible world semantics*). Logiczne podstawy tego rodzaju modeli normatywnych i predykcyjnych przedyskutowałem szerzej w [4].

Nie wydaje mi się, żeby w pracach poświęconych logikom niemonotonicznym i modalnym zwrócono uwagę na fakt, że nauki fizyczne i ekonomiczne posługują się już od wielu pokoleń reprezentacją przestrzeni stanów i standardowym wnioskowaniem matematycznym, analizując zachowanie się systemów w czasie, i że nie natknięto się jak dotychczas na żadne szczególne trudności związane z zachowaniem prawdy (ang. *truth maintenance*). Czasowe lub przestrzenne uwarunkowanie wartości zmiennych opisuje się zwięźle i ściśle za pomocą notacji indeksowej.

Pozostając w zgodzie z naszymi spostrzeżeniami na temat systemów wnioskowania z poprzedniego punktu, dochodzimy do wniosku, że siła systemów przeszukiwania przestrzeni rozwiązań leży przede wszystkim w wykorzystaniu efektywnych operatorów przejścia, które służą jako makroreguły wnioskowania. Przy założeniu, że operatory zostały właściwie zdefiniowane, wnioskowanie przeprowadzone za ich pomocą jest ścisłe (w tym samym sensie, w jakim matematyka stosowana jest ścisła) i nie wymaga bardziej formalnego aparatu logicznego.

MODEL A RZECZYWISTOŚĆ

Pierwsze badania nad wykorzystaniem logiki modalnej w sztucznej inteligencji koncentrowały się głównie na zagadnieniach wnioskowania o akcjach. McCarthy i Hayes [3] usiłowali skonstruować rachunek przyczynowości zawierający predykaty typu *CANCAUSE (A, C)*, gdzie *A* jest akcją, a *C* – jej konsekwencją. Okazało się jednak, że logika modalna tego rodzaju, uzupełniona o odpowiednie (ani zbyt słabe, ani zbyt mocne) reguły wnioskowania, jest bardzo zawodna [5].

Logika przyczynowości i możliwości

Podstawowa trudność w stworzeniu logiki, która mogłaby z powodzeniem posługiwać się modalnościami możliwości i przyczynowości, leży w regułach składania. Jesteśmy skłonni – przez analogię do rachunku predykatów – wprowadzić regułę, która pozwoliłaby nam przy założeniu, że *„A umożliwia B"* i *„A umożliwia C"*, wywnioskować, że *„A umożliwia B i C"*. Lecz z pewnością z tego, że *„15 tysięcy dolarów umożliwia zakup samochodu"* i *„15 tysięcy dolarów umożliwia zakup jachtu"*, nie możemy wywnioskować, że *„15 tysięcy dolarów umożliwia zakup samochodu i jachtu"*.

Z łatwością można skonstruować wiele podobnych paradoksów [5]. Są one na ogół rezultatem braku niezależności zdarzeń. Gdy zdarzenia są wzajemnie zależne, to prawa składania o zwykłej mocy generują paradoksy. Jeśli jednak takich praw nie ma, to logika jest zbyt słaba, by móc wyciągać wnioski niezbędne w rozwiązywaniu problemów z uwzględnieniem przyczynowości i możliwości.

Trudności tych możemy uniknąć traktując pojęcia modalne jako heurystyki (nie uwzględnia się wówczas heurystyki bezpośrednio w systemie wnioskowania logicznego). Dla przykładu, heurystyki użyte w metodzie analizy środków i celów systemu *GPS* realizują funkcje rachunku przyczynowości i możliwości. Procedura przeszukująca systemu *GPS* jest oparta na niejawnej przesłance, że jeśli aktualna sytuacja różni się od sytuacji docelowej cechami *A*, *B*, *C*..., to do sytuacji docelowej można dojść eliminując w pewnej kolejności rozbieżności *A*, *B*, *C*... Oczywiście przesłanka ta jest fałszywa, jeśli macierzy zależności między rozbieżnościami i operatorami nie można przekształcić do postaci trójkątnej. Macierz trójkątną możemy uzyskać tylko wtedy, gdy w logice modalnej obowiązuje odpowiedni aksjomat składania; innymi słowy tylko wtedy, gdy akcje są niezależne.

Zaletą postępowania heurystycznego jest to, że nie konstruuje się zdań, które mogą okazać się nieważne, lecz że rozważamy jedynie możliwości, które mogą się sprawdzić lub nie. Oczywiście oprócz analizy środków i celów istnieją inne heurystyki przeszukiwania przestrzeni rozwiązań. Przewaga podejścia heurystycznego nad logiką modalną dotyczy również innych heurystyk przeszukiwania.

Wnioskowanie niemonotoniczne

Najnowsze badania w dziedzinie logiki niemonotonicznej zajmują się na ogół innymi zagadnieniami niż te, które zostały poruszone w naszej dotychczasowej dyskusji i koncentrują się w szczególności na zagadnieniu, które moglibyśmy nazwać *„wnioskowaniem na podstawie niedostatecznych dowodów"*. Istotą wnioskowania tego rodzaju jest konstruowanie nowych zdań bez bezpośredniego dowodzenia ich zgodności z pozostałymi zdaniami, w taki jednak sposób, aby nie pojawiła się później konieczność ich zanegowania. Za przykład mogłoby posłużyć zdanie: *„Jeśli nie jest ci znany jakikolwiek powód, dla którego nie mógłbyś pojechać do X autobusem, to możesz pojechać autobusem"*.

Dlaczego jest wygodnie posługiwać się takimi regułami, dowiemy się nieco później. Reguły tego rodzaju wymagają kilku nowych mechanizmów. Po pierwsze, musi istnieć procedura decydująca o tym, co jest dostatecznym dowodem na to, żeby móc wypowiedzieć jakieś zdanie, lub też dostatecznym dowodem na to, żeby zdanie to nie zaprzeczało innym zdaniom. Po drugie, systemy tego rodzaju nie wykluczają możliwości, że coś, co w danej chwili jest niesprzeczne (a więc dające się stwierdzić), może później stać się sprzeczne. Dlatego też niezbędna jest procedura usuwająca napotykane sprzeczności. Można oczywiście

skonstruować wiele różnych systemów wnioskowania wprowadzając modyfikacje do tych dwóch procedur.

John McCarthy [2] posłużył się zagadką o misjonarzach i ludożercach w celu uzasadnienia niezbędności wprowadzenia wnioskowania modalnego i niemonotonicznego. Zauważa on, że tę zagadkę opisuje się formalnie zazwyczaj za pomocą przestrzeni stanów w sposób zbliżony do przedstawionego w tym artykule. Następnie podaje kilka zastrzeżeń [2, s. 29–30]:

„Nie zajmujemy się teraz heurystykami problemu, lecz raczej poprawnością rozumowania, które przeprowadza nas od sformułowania problemu w języku angielskim do... [formalnej]... reprezentacji przestrzeni stanów. Takie rozumowanie mógłby wykonać ogólnie inteligentny program komputerowy. Oczywiście znane są powszechnie trudności z rozumieniem języka angielskiego przez komputery, załóżmy więc, że angielskie zdania opisujące problem zostały już przetłumaczone bezpośrednio na zdania logiki pierwszego rzędu. Poprawność... [formalnej]... reprezentacji tych zdań nie jest jednak zwyczajną konsekwencją logiczną, a to z dwóch powodów.

Po pierwsze, nic nie zostało powiedziane o właściwościach łodzi oraz o tym, że wiosłowanie przez rzekę nie zmienia liczby misjonarzy i ludożerców ani też pojemności łodzi. W rzeczywistości nie powiedziano też, że sytuacje ulegają zmianie pod wpływem akcji. Fakty te wynikają z wiedzy o świecie (czyli ze zdrowego rozsądku), a więc wyobraźmy sobie, że ta wiedza o świecie, lub przynajmniej istotna jej część, została również wyrażona w logice pierwszego rzędu.

Drugi powód, dla którego nie możemy wydedukować właściwości ...[formalnej]... reprezentacji jest głębszy. Wyobraźmy sobie, że zadaliśmy zagadkę komuś, kto po chwili zastanowienia sugeruje, żeby pójść pół mili w górę rzeki i przejść przez most.

– Jaki most? – pytamy.

W sformułowaniu problemu nie ma mowy o żadnym moście.

Nasz mędrek na to:

– Ale sformułowanie nie mówi, że nie ma mostu.

A więc modyfikujemy sformułowanie, aby wykluczyć mosty, zadajemy zagadkę ponownie, a na to mędrek proponuje polecieć helikopterem...

Poirytowani odpowiedziami naszego mędrka nie powinniśmy jednak ulec pokusie uzupełnienia sformułowania problemu dodatkowymi informacjami o tym, że rzekę można przekroczyć jedynie łodzią i że łodzi nie może przytrafić się nic złego. Człowiek nie potrzebuje takiego specjalnego zawężenia problemu; w rzeczywistości jedynie pewnym sposobem osiągnięcia tego celu byłoby skonstruowanie [formalnej] reprezentacji w języku angielskim. Wolimy jednak uniknąć podawania nadmiernej liczby warunków i dążymy raczej do uzyskania [formalnej] reprezentacji w drodze rozumowania zdroworozsądkowego, tak jak to czynią zwykle ludzie".

Jeśli wrócimy na chwilę do naszego przykładu z wodą i alkoholem, to zauważymy, że w algebraicznym problemie występują te same trudności, na które zwrócił uwagę McCarthy analizując zagadkę o misjonarzach i ludożercach. W sformułowaniu problemu brakowało kilku równań istotnych z punktu widzenia rozwiązania problemu i nie było wiadomo z góry, czy dane i wywnioskowane równania będą stanowić wszystkie niezbędne relacje ograniczające rozpatrywaną sytuację. Fakt *„że wiosłowanie przez rzekę nie zmienia liczby misjonarzy i ludożerców ani też pojemności łodzi"* jest założeniem zachowawczym dokładnie tego samego rodzaju, jakim jest założenie, że całkowita objętość cieczy będzie sumą pierwotnej objętości cieczy i objętości dolanej wody.

Wiedza zdroworozsądkowa

McCarthy z pewnością ma rację, gdy zauważa, że osoba rozwiązująca jakiś problem musi posłużyć się wiedzą o świecie, nie występującą jawnie w sformułowaniu problemu. Czy ma jednak rację zakładając, że ta wiedza powinna być dostarczona tak, jak w przykładzie z algebry, w postaci jawnych zdań logiki? Wiedza jest potrzebna niezależnie od tego, czy przyjmujemy punkt widzenia przeszukiwania przestrzeni rozwiązań, czy też punkt widzenia procesu wnioskowania. Jak jednak opisać taką wiedzę za pomocą reprezentacji przestrzeni stanów w wypadku zagadki o misjonarzach i ludożercach, i skąd tę wiedzę wziąć?

Typowa reprezentacja tej zagadki, skonstruowana za pomocą języka przetwarzania list, mogłaby składać się z list właściwości dla obu brzegów rzeki. Wartością właściwości *„misjonarze"* mogłaby być albo liczba misjonarzy, albo lista misjonarzy na danym brzegu rzeki. Operator przejścia zmieniałby wartość każdej z tych właściwości dla obu brzegów rzeki, nie zmieniając przy tym całkowitej liczby misjonarzy, ludożerców i łodzi. Testy zapewniające poprawność przejścia mogłyby być zawarte w samych operatorach lub też mogłyby być niezależnymi procedurami.

Dzięki takiemu zabiegowi założenia zachowawcze o misjonarzach, ludożercach i łodziach można ukryć w reprezentacji, nie określając ich za pomocą zdań logiki. System nie dochodzi więc sam do wniosku, że pewne wielkości muszą być zachowane, ani nie dysponuje żadnymi zdaniami, które mogłyby być użyte jako przesłanki w takim wnioskowaniu. Cała wiedza jest wbudowana w strukturę operatora przejścia, który sam zachowuje niezbędne wielkości.

Jak już pokazano, punkt widzenia procesu wnioskowania można przywrócić stosując pewną procedurę, której używa się także w logice formalnej. W logice istnieje na ogół możliwość dodania do systemu nowych aksjomatów albo nowych reguł wnioskowania. Jeśli uznać, że operatory przejścia w reprezentacji stanów są regułami wnioskowania zachowującymi prawdę, to faktycznie postuluje się tutaj dodanie takiej reguły, a nie aksjomatu.

Analogię tę (mimo że daje się ją formalnie obronić) należy stosować z ostrożnością. Założenia zachowawcze zawarte w tym, co nazywa się obecnie regułą wnioskowania, są – jak już podkreślono – założeniami empirycznymi, a nie logicznymi. Wszystkie twierdzenia co do słuszności tych założeń muszą więc być oparte na faktycznej wiedzy, a nie na czystej argumentacji logicznej. Aby jednak nie ulec mirażowi *„oczywistości"* zasad zachowania, należy pamiętać o pewnych ich właściwościach.

Po pierwsze, jak wykazał Piaget, dzieci nie rodzą się z przekonaniem, że przedmioty nie przestają istnieć, gdy chwilowo znikają z pola widzenia. Dzieci nabierają takiego przekonania w wyniku nauki i (lub) dojrzewania. W tradycji piagetańskiej istnieje bogata literatura na temat przyswajania zasad zachowania. Po drugie, rodzaj ludzki jako całość przez długi czas dochodził do bardziej wyrafinowanych zasad zachowania, takich jak zachowanie masy czy zachowanie energii. Najwidoczniej zasady te nie były wcale takie oczywiste. Odkrycie ich kosztowało naukowców wiele potu i łez, jeśli nie krwi. Po trzecie, nie wszystkie wielkości podlegają prawom zachowania. Widzieliśmy już, że nawet proste założenia o zachowaniu objętości mieszaniny cieczy nie zawsze są poprawne empirycznie.

Powróćmy jeszcze raz do wniosków z naszych poprzednich rozważań. Wiedza zdroworozsądkowa, o którą należy rozszerzyć reprezentację problemu, jest wiedzą empiryczną, specyficzną dla dziedziny danego problemu i podlegającą modyfikacji wraz ze zmianą wiedzy empirycznej. Ale wiedza ta nie musi być zadana w postaci jawnych zdań logiki, lecz może być zawarta niejawnie w operatorach przejścia, symulujących efekty akcji i zdarzeń w modelowanym świecie. Nagina się nieco rzeczywistość interpretując te operatory jako *„reguły wnioskowania"*, gdyż zawierają one w sposób niejawny zarówno założenia empiryczne, jak i logiczne.

Przyjmując w rozwiązywaniu problemów punkt widzenia procesu wnioskowania, można łatwo ulec pokusie jawnego przedstawienia założeń problemu za pomocą zdań logiki. Doświadczenie zdobyte w trakcie rozwiązywania problemów z wykorzystaniem reprezentacji przestrzeni stanów sugeruje, że bardziej celowe mogłoby być ukrycie wiedzy (zarówno empirycznej, jak i logicznej) w operatorach.

Wnioskowanie zdroworozsądkowe

Rozważmy teraz drugi problem poruszony w cytacie McCarthy'ego. Skąd osoba rozwiązująca problem wie, że nie ma innych operatorów prócz tych, które zna? Jakie reguły wnioskowania pozwalają założyć, że w zagadce o misjonarzach i ludożercach nie istnieje most położony o pół mili w górę rzeki?

Pomimo tego, że McCarthy mówi tutaj o *„wnioskowaniu"*, a nie o *„wiedzy"*, trudność ta nie jest w rzeczywistości różna od tej, którą

omówiliśmy uprzednio. To czy most istnieje, czy też nie, nie jest sprawą logiki – jest to po prostu fakt. Jeśli naprawdę istnieje wygodny most, który nie został uwzględniony w formalnej reprezentacji problemu, to reprezentacja ta daje faktycznie niepoprawny opis zadania.

Aby utworzyć poprawną reprezentację problemu, z pewnością nie musimy opisywać za pomocą zdań logiki tego, czego nie ma – tzn. że nie ma mostu, helikoptera itd. Matka Hubbard[1] nie musiała wyliczać wszystkich rzeczy, których nie było w szafie; wystarczyło po prostu zobaczyć tylną ścianę szafy, by stwierdzić, że szafa jest pusta. Niezależnie od tego, czy przyjmujemy punkt widzenia wnioskowania, czy też przeszukiwania przestrzeni rozwiązań, o sytuacjach określonego problemu można wnioskować poprawnie posługując się reprezentacją, która jawnie mówi, co jest prawdziwe w danej sytuacji, i która nic nie mówi o tym, co nie jest prawdziwe. A to, co jest prawdą a co nie, dotyczy faktu i nie jest sprawą logiki.

Ograniczona racjonalność

Wiele spośród trudności związanych z wnioskowaniem zdroworozsądkowym znika, gdy zauważymy, że w rzeczywistości nie mamy wcale do czynienia ze światem, w którym problemy zawsze mogą być rozwiązane, i to w dodatku optymalnie. Mamy raczej do czynienia z różnymi odmianami ograniczonej racjonalności, do której skłonne są istoty ludzkie.

„Wnioskowanie" potrzebne w rozwiązywaniu problemów jest na ogół indukcyjne, a nie dedukcyjne. Naszym celem jest znalezienie ciągu akcji, które przekształcą początkową sytuację na sytuację spełniającą warunki rozwiązania. Najczęściej nie wymaga się, aby rozwiązanie było jednoznaczne lub, co za tym idzie, spełniało określone warunki optymalności. Sformułowanie zagadki o misjonarzach i ludożercach nie zawiera żądania, żeby przekroczyć rzekę jak najszybciej lub z jak najmniejszym wysiłkiem ze strony misjonarzy. Rozwiązanie można oczywiście uznać za dowód, że podejmując określone akcje osiągamy określony cel, lecz nie istnieje taki proces dedukcyjny, za pomocą którego można by dowieść, że akcje te **muszą** być podjęte, aby osiągnąć ten cel.

Powyższa uwaga znacznie osłabia nasze zaniepokojenie co do możliwości istnienia mostów w górze rzeki. Fakt, że nie znamy innych alternatyw, nie uniemożliwia rozwiązania problemu. Ludzkość rozwinęła wiele efektywnych systemów transportowych, zanim nauczyła się budować i latać samolotami. Nasza formalna reprezentacja obejmuje po prostu te alternatywy, których jesteśmy świadomi i do których mamy dostęp. Z tego, że nie udało się nam uwzględnić tych możliwości w reprezentacji, nie wynika – ani logicznie ani empirycznie – że inne możliwości nie istnieją.

Jest to szczęśliwy zbieg okoliczności, gdyż w przeciwnym razie moglibyśmy rozwiązywać tylko nieliczne problemy praktyczne. Już dawno zauważono [6], że istoty ludzkie idą zwykle na kompromis, poszukując satysfakcjonujących, a nie optymalnych rozwiązań swych problemów. Jedną z głównych przyczyn takiego stanu rzeczy jest to, że ludzie rzadko mają pewność, czy stworzone przez nich reprezentacje problemów obejmują wszystkie alternatywy. Inną przyczyną jest to, że nawet w ramach określonej reprezentacji problemu znacznie łatwiej jest znaleźć **jakieś** rozwiązanie niż znaleźć jednoznaczne, najlepsze rozwiązanie.

LITERATURA

[1] Fikes R.E., Nilsson N.J.: STRIPS – A new approach to the application of theorem proving to problem solving. Artificial Intelligence, Vol. 2, pp. 189–208, 1971
[2] McCarthy J.: Circumscription – a form of non-monotonic reasoning. Artificial Intelligence, Vol. 13, pp. 27–40, 1980
[3] McCarthy J., Hayes P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, American Elsevier, New York, 1969
[4] Simon H.A.: Models of Discovery. Reidel, Dordrecht, 1977
[5] Simon H.A.: On reasoning about actions. Pp. 414–430, Representation and Meaning, Simon H.A., Siklossy L. (Eds.), Prentice-Hall, Englewood Cliffs (NJ), 1972
[6] Simon H.A.: Administrative Behaviour. Macmillan, New York, 1974.

Tłumaczył i opracował
MAREK MACHURA

[1] Postać z tradycyjnego angielskiego wierszyka dla dzieci (przyp. tłum).

JANUSZ RYBNIK
JERZY SOLAK
Instytut Informacji Naukowej Technicznej i Ekonomicznej
Warszawa

POLSKIE TOWARZYSTWO INFORMATYCZNE

Stacje robocze Sun-3 (2)

Oprogramowanie

W drugiej części artykułu omówiono oprogramowanie systemowe, środowiska programowe oraz oprogramowanie sieciowe i graficzne.

STRUKTURA OPROGRAMOWANIA

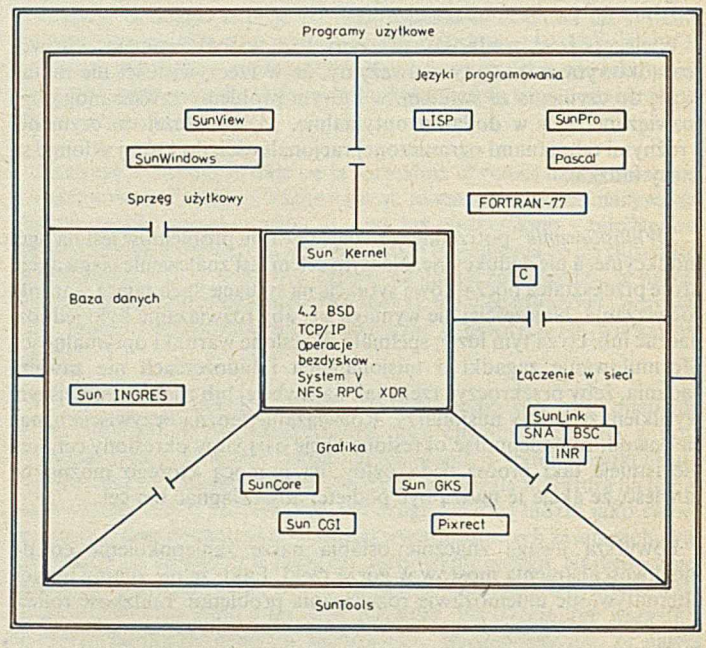
Oprogramowanie komputerów Sun zostało zaprojektowane w taki sposób, by było wygodne w użyciu dla szerokiego grona użytkowników. Podstawą oprogramowania jest system operacyjny Unix 4.2BSD. Firma Sun Microsystem w kooperacji z firmą AT&T doprowadziła do połączenia systemów Unix 4.2BSD i Unix System V. Mówiąc ściślej, firma Sun zaadaptowała wersję 4.2BSD jako podstawę dla swojego systemu operacyjnego, który stale rozwija i rozbudowuje o nowe możliwości (np. o grafikę, mysz, system okien).

Unix jest wielozadaniowym, wielodostępnym systemem operacyjnym prostym w użyciu i łatwym do rozbudowy. Jego zalety spowodowały, że stał się standardem w środowiskach akademickich, a jednocześnie zdominował rynek mikrokomputerowy.

Popularność Unixa można wytłumaczyć trzema następującymi zaletami:
- jest systemem łatwym do adaptacji dla różnych komputerów, daje dużą niezależność od sprzętu i jednocześnie umożliwia łatwe przenoszenie oprogramowania między różnymi komputerami;
- zapewnia bardzo bogate oprogramowanie narzędziowe dla bardzo szerokiego grona użytkowników;
- ma prostą, elegancką strukturę i mimo iż był projektowany już prawie 20 lat temu, nic nie stracił ze swej wartości w dobie mikrokomputerów.

Słabości Unixa można sprowadzić do następujących zarzutów:
- komunikaty błędów są zbyt lakoniczne i niewiele mówiące, system poleceń jest niespójny i zbyt zwarty, a niektóre polecenia mają nieczytelne nazwy, np. *grep*, *awk*;
- Unix zakłada świadomość działania użytkownika, np. zamazuje stary plik, jeżeli jest otwierany nowy z tą samą nazwą;
- wymaga dużo pamięci; w konfiguracji z dyskiem 71 MB Unix zajmuje od 54 do 59 MB, tzn. że dla użytkownika zostaje od 12 do 17 MB (w obliczeniach tych nie uwzględniono oprogramowania podstawowego, np. systemu okien, kompilatorów Pascala i Fortranu, pakietów graficznych, gier, programów demonstracyjnych i podręczników, które w wersji minimalnej zajmuje ok. 10 MB);
- koncepcja stosowania makropoleceń, w założeniu bardzo wygodna i użyteczna, spowodowała powstanie tylu różnych wersji systemu, że przejście z jednej instalacji na drugą staje się czasami trudne do wykonania.

Sunowska wersja Unixa jest dostarczana z zestawem następujących kompilatorów: Fortran 77, Pascal C i Assembler, a ponadto jako opcja Common Lisp. Podstawą środowiska programowego, ułatwiającego przygotowywanie i testowanie programów, jest pakiet *SunPro*, współpracujący z systemem okien *SunView*. Otoczenie bazodanowe jest oparte na modelu relacyjnym *Sun Ingres* i *SunView*, co daje dużą elastyczność i niezależność dostępu do danych. *SunLink* umożliwia łączenie stacji roboczych w sieci homogeniczne (tzn. łączenie komputerów tego samego typu), jak i heterogeniczne (np. łączenie z IBM PC). Bardzo bogate oprogramowanie graficzne zawiera trzy standardowe pakiety: *SunCore*, *SunCGI* i *SunGKS*. Strukturę architektury oprogramowania stacji Sun przedstawiono schematycznie na rys. 1.



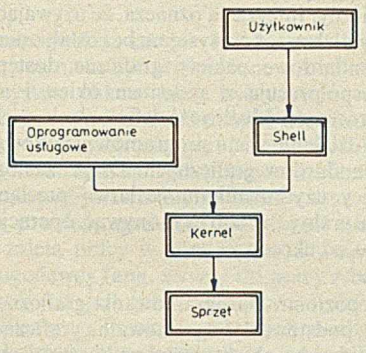
Rys. 1. Architektura oprogramowania stacji Sun-3
NFS (ang. network file system) – sieciowy system plików,
TCP/IP (ang. transmission control procedures, interconnection protocol) protokoły komunikacyjne,
RPC (ang. remote procedure call) wywoływanie odległych procedur sieci,
XDR (ang. external data representation) zewnętrzny obraz danych,
Pixrect (ang. pixel rectangle) – biblioteka procedur graficznych niskiego poziomu

OPROGRAMOWANIE SYSTEMOWE

Ponieważ Unix 4.2BSD został rozwinięty i dostosowany do potrzeb stacji roboczych, a ponadto wzbogacony o możliwość korzystania z oprogramowania systemu V, użytkownicy i programiści mogą równocześnie wykorzystywać zalety obu tych systemów. Dzięki temu, że system operacyjny jest odpowiedzialny za kontrolę procesów i zarządzanie zasobami, przy jednoczesnym odseparowaniu użytkownika od poziomu sprzętowego, projektanci oprogramowania są zwolnieni od bezpośredniego współdziałania ze sprzętem i od znajomości stanu innych procesów, choć z drugiej strony ogranicza to nieco swobodę ich działania. Niemniej Unix jest doskonałą bazą do konstruowania oprogramowania systemowego.

Strukturę Unixa przedstawiono na rys. 2. Jądro (ang. *kernel*) bezpośrednio zarządza sprzętem, stanowiąc właściwy system operacyjny – w odróżnieniu od powłoki (ang. *shell*) i oprogramowania użytkowego, które są zwykłymi programami bez żadnych specjalnych przywilejów. Podstawowe zadania jądra można podzielić na cztery grupy:
- zarządzanie procesami,
- zarządzanie pamięcią,
- obsługa wejścia-wyjścia,
- zegar.

Całe otoczenie procesów i system plików są zarządzane przez jądro, a inicjowanie procesów jest wykonywane przez użytkownika z poziomu powłoki. W Unixie firmy Sun są dostępne dwie wersje powłoki: *C-Shell* (nazwa oznaczająca podobieństwo składni do języka C) i *Bourne Shell* (od nazwiska autora). Obie mogą być używane jako interakcyjne interpretery poleceń albo jako języki programowania, aczkolwiek zwykle *C-Shell* jest używany jako interpreter, a *Bourne Shell* jako język programowania. Oprócz tego użytkownik działający w sieci może korzystać z tzw. *Remote Shell* – zdalnego interpretera poleceń w innym systemie w sieci. Każde polecenie wydane przez użytkownika jest analizowane i wykonywane przez powłokę jako niezależne zadanie pierwszo- lub drugoplanowe. Powłoka zajmuje się przydzielaniem standardowych plików procesom, wiązaniem procesów przez pliki, czyli tworzeniem tzw. potoków (ang. *pipe*), wykonywaniem procedur rejestrowania (ang. *login/logoff*). Mechanizmy definiowania zmiennych i polecenia sterujące *if...then...else*, *while*, *for* itp. pozwalają na wykorzystywanie zarówno *C-Shell* jak i *Bourne Shell* do pisania programów, które mogą być składowane w plikach dyskowych.



Rys. 2. Struktura systemu Unix

Oprogramowanie usługowe stacji Sun-3, dostarczane wraz z systemem operacyjnym, można podzielić na dziewięć następujących grup:
- **programy podstawowe** – podstawowe moduły systemowe Unixa, realizujące polecenia proste;
- **edytory** – wierszowe (*ed*) i ekranowe (*vi*, *ex*);
- **narzędzia programowe** – kompilatory, asembler, linker, program uruchomieniowy, optymalizator, profiler, kompilator kompilatorów (czyli generator parserów na podstawie opisu języka), narzędzia ułatwiające utrzymywanie porządku przy realizacji dużych projektów;
- **filtry** – czyli programy odczytujące standardowe wejście, transformujące dane i wysyłające je na standardowe wyjście;
- **formatery** – tj. narzędzia (*nroff* i *troff*) do formatowania wydruków dokumentów, artykułów i książek;
- **programy komunikacyjne**;
- **oprogramowanie baz danych**;
- **system okien**;
- **oprogramowanie graficzne**.

Użytkownicy komputerów Sun mogą ponadto korzystać z ofert zebranych w wydawnictwie Catalyst. Jest to bardzo bogaty katalog programów użytkowych (ponad 600 pozycji w 1987 r.), podzielonych na 16 działów (biomedycyna, architektura, zarządzanie, CAD/CAE, mechanika, działalność edytorsko-publikacyjna, bazy danych, grafika, sztuczna inteligencja, statystyka i matematyka, prace biurowe itp.).

Oprogramowanie systemowe, jego funkcjonalność i bogactwo, możliwości graficzne i sieciowe, okienkowy dostęp użytkownika do zasobów stawiają stacje Sun w rzędzie najlepszych stacji roboczych. W dokumentacji podkreśla się, że jest to zasługa systemu operacyjnego Unix 4.2BSD, oczywiście po wielu latach pracy nad jego rozwojem.

ŚRODOWISKO PROGRAMISTY

Środowisko programisty stacji roboczych Sun jest skonstruowane zgodnie z najlepszymi tradycjami systemu Unix. Zapewnia łatwe dołączanie nowych programów, szybkie budowanie nowego oprogramowania i łatwość korzystania ze wszystkich dostępnych możliwości. Oprócz wymienionych programów wchodzących standardowo w skład tego środowiska są dostępne na rynku kompilatory języków Prolog, Smalltalk, Ada, APL, Basic, Cobol i inne. Kompilatory języka C, Fortranu 77 i Pascala wytwarzają uniwersalny kod wynikowy i uniwersalną tablicę symboli. Dzięki temu programy napisane w jednym z tych języków mogą wywoływać procedury (własne użytkownika lub biblioteczne) napisane w innym języku. Pozwala to także na korzystanie ze wspólnego programu uruchomieniowego. *Dbxtool* – symboliczny program uruchomieniowy z dostępem okienkowym i myszą – umożliwia szybkie i łatwe uruchamianie programów w cyklu redagowanie–kompilowanie–testowanie. Ponadto, pakiet *SunPro* dostarcza narzędzi do pełnej obsługi dużych projektów (kontrola programów źródłowych, zarządzanie zasobami i łączność między poszczególnymi elementami projektu), a narzędzia graficzne pozwalają na przejrzystą prezentację wyników.

ŚRODOWISKO UŻYTKOWNIKA

W celu ułatwienia współpracy użytkowników ze stacją roboczą typu Sun-3, wraz z systemem operacyjnym jest dostarczany pakiet *SunView* (ang. *Sun Visual Integrated Environment for Workstations*). Składa się on z dwóch części:
- sprzęgu użytkownika, dostępnego przez pakiet graficzny *SunTools* (system okienkowy, w którym jednocześnie można otworzyć wiele niezależnych od siebie okien, a w każdym z nich – uruchomić samodzielny proces);
- *SunGuide* (ang. *Sun General User Interface Design Environment*) – programowego sprzężenia, składającego się z biblioteki podprogramów do budowy okienkowych systemów współpracy z komputerem.

Użytkownik komunikuje się z pakietem *SunTools* przy użyciu monitora i myszy. Dzięki systemowi okien może on łatwo uruchamiać i obserwować jednocześnie wiele współbieżnych procesów. Okna zajmują różne obszary ekranu i mogą na siebie zachodzić. Manewrując myszą można je przesuwać po ekranie, zmieniać ich wymiary i wreszcie tworzyć nowe okna i likwidować lub zamknąć już istniejące. Po zlikwidowaniu okno wraz ze swoim procesem przestaje istnieć, a po zamknięciu proces zostaje zawieszony, a okno jest zamieniane na ikonę symbolizującą rodzaj zamkniętego okna. Okno zamknięte może być w dowolnym momencie ponownie otwarte.

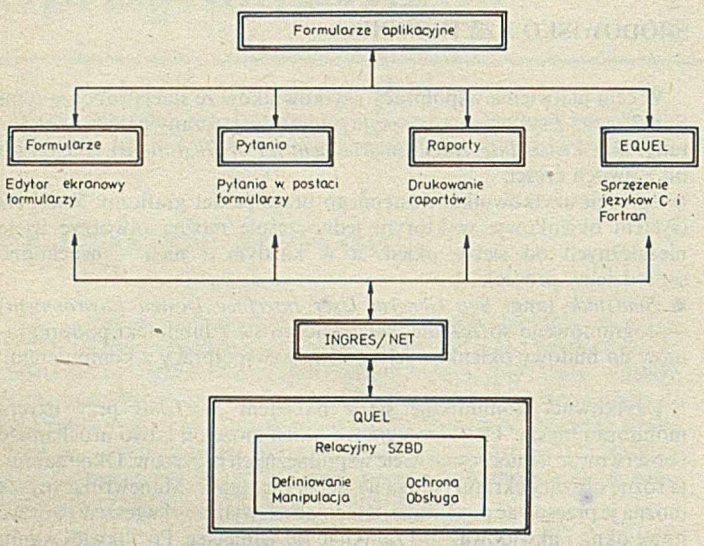
SunView zawiera następujące standardowe okna:
- *MailTool* – program pozwalający na tworzenie, przechowywanie, wyświetlanie i wysyłanie przesyłek pocztowych w obrębie sieci komputerowej;
- *ShellTool* – okno dostępu do standardowego interpretera poleceń systemu Unix;
- *CommandTool* – okno pozwalające na wydawanie poleceń;
- *TextEditor* – edytor tekstowy współpracujący z myszą;
- *DefaultEditor* – edytor do ustawiania parametrów systemu Unix;
- *IconEditor* – edytor do rysowania i projektowania kursorów i ikon;
- *DbxToll* – symboliczny program uruchomieniowy;
- *ClockTool* – zegar i dątownik;
- *PerfMeter* – program przedstawiający graficznie wykorzystanie zasobów komputera (zajętość pamięci, obsługa *we-wy*, obciążenie CPU itp.);
- *FontEdit* – edytor do projektowania i tworzenia znaków graficznych;
- *GraphicsTool* – sprzężenie graficzne;
- *Setup* – program ustawiający i sprawdzający parametry dostępnej konfiguracji;
- *TtyTool* – emulator terminali ANSI.

SunView umożliwia definiowanie okien, zarządza nimi i kontroluje współpracę z użytkownikami. Każde okno może zawierać wiele podokien, z którymi mogą być związane aktywne procesy. Użytkownik może pobierać informacje z dowolnych podokien, korzystając z procesów uruchomieniowych w innych oknach lub podoknach. To samo może wykonać, korzystając z mechanizmu ładowania i rozładowywania bufora, korzystając z myszy lub bezpośrednio przez komunikację między niezależnymi procesami. Wszystkie standardowe okna mogą być łączone z oknami tworzonymi przez użytkownika.

BAZY DANYCH

Firma Sun zaadaptowała relacyjny system zarządzania bazą danych INGRES, tworząc w pełni zintegrowane z Unixem środowisko dla baz danych. Sun INGRES jest systemem relacyjnym z nieproceduralnym językiem zapytań QUEL. Programowe sprzężenie pozwala na „zanu-

rzanie" formuł języka QUEL w Fortranie 77 i w języku C (EQUEL – Embedded QUEL). Sprzężenie systemu INGRES z SunView umożliwia interakcyjną współpracę z bazą danych. Ponadto, użytkownik może zadawać pytania w trybie wsadowym. Dzięki zastosowaniu systemu okien i narzędzi graficznych Sun INGRES pozwala formułować pytania do bazy danych w postaci formularzy. Formularze mogą być również używane do wyprowadzania wyników wyszukiwania. Można się przy tym posługiwać standardowymi formularzami bibliotecznymi lub definiować własne formularze przy użyciu specjalnego edytora. Pakiet *INGRES/NET* umożliwia łączenie baz w sieci i dostęp do rozproszonych baz danych. Schemat systemu baz danych INGRES przedstawiono na rysunku 3.



Rys. 3. Schemat systemu obsługi baz danych INGRES

SIECI

Stacje robocze Sun można łączyć zarówno w sieci jednorodne (tylko komputery Sun), jak i niejednorodne (np. łączenie przez *SunLink* z IBM). Właściwa dla stacji Sun architektura sieci lokalnej jest oparta na normie Ethernet, choć w ogóle można je łączyć w sieci trzech rodzajów:

- *Ethernet* – lokalna sieć składająca się z komputerów połączonych bezpośrednio specjalnym kablem współosiowym, umożliwiająca komunikację między komputerami w czasie rzeczywistym; w stacjach Sun używa się standardowych protokołów *TCP/IP*.
- *Sieć UUCP* – połączenie komputerów przez sieć telefoniczną; UUCP (ang. *Unix to Unix Copy Program*) obejmuje całe Stany Zjednoczone i wiele innych krajów na świecie; nie jest to sieć czasu rzeczywistego, co oznacza, że nie można przez nią współpracować z komputerem na prawach normalnego użytkownika, ale można przesyłać informacje między komputerami.
- *DARPANET (ARPANET/MILNET) – Defense Data Network (DARPANET)* lub *Advanced Research Projects Agency (ARPANET)* obejmuje USA i kilka ośrodków europejskich; użytkownicy dołączeni bezpośrednio do tej sieci za pomocą kabli lub łączy mikrofalowych mogą współpracować w czasie rzeczywistym z innymi działającymi w niej komputerami.

Między komputerami Sun a innymi systemami są możliwe dwa rodzaje połączeń:

- przesyłanie plików, zdalne wykonywanie poleceń, obsługa terminala wirtualnego i emulacja terminala dla systemów akceptujących *TCP/IP;*
- dostęp do plików rozproszonych (*NFS* – ang. *network file system)* i rozproszone przetwarzanie (*NFS i RPC* – ang. *remote procedure calls); NFS* może łączyć wszystkich użytkowników dołączonych do sieci za pomocą satelity, łączy mikrofalowych lub światłowodów, a *RPC* może być wywoływane z dowolnego miejsca w sieci z dowolnej odległości. Ponadto, przez protokół *YP* (ang. *yellow pages)*, określający rozproszoną bazę danych dotyczącą konfiguracji sieci, może być zapewnione sterowanie i zarządzanie całą siecią. Każda zmiana w konfiguracji sieci jest odnotowywana w *YP*, a informacja o niej jest rozsyłana przez sieć tak, by każdy użytkownik był o tej zmianie uprzedzony.

Przez pakiet *SunLink*, stacje robocze Sun mogą być łączone w sieci z innymi komputerami, używającymi różnych protokołów. Obecnie *SunLink* obejmuje następujące produkty: *SNA* 3270, *BSC RJE* 2780/3780 *HASP* i *INR* (ang. *inter network router*). Jednocześnie jest opracowywany protokół *X.25* i protokoły *ISO*, które mają zostać włączone w najbliższym czasie do rodziny produktów *SunLink*. *SNA* 3270 i *BSC RJE* zapewniają zgodność pakietu *SunLink* z produktami rodziny 360/370 i popularnymi komputerami osobistymi firmy IBM, a także z innymi mikro- i minikomputerami oraz procesorami tekstowymi.

GRAFIKA

Szybki rozwój sprzętu i oprogramowania w ostatnich latach spowodował wzrost zainteresowania grafiką, jej możliwościami i dostępnością dla szerokiego grona użytkowników. Środowisko graficzne komputerów rodziny Sun-3 integruje normy graficzne (*SunCGI*, *SunCore*, *SunGKS*), rozwinięty system okien, sieciowy system plików i bardzo szybki sprzęt graficzny. Integracja oznacza, że używając grafiki, można korzystać ze wszystkich zasobów systemu bez osłabiania jego sprawności. Wszystkie standardowe pakiety graficzne dostępne w stacjach rodziny Sun-3 współpracują z systemem okien i mogą korzystać z graficznego koprocesora. Możliwość definiowania wielu okien pozwala na jednoczesne działanie wielu programów użytkowych, korzystających z różnych standardów graficznych. Dzięki zastosowaniu myszy, drążka i rysownicy użytkownik może łatwo przełączać się między procesami i w naturalny sposób wykonywać operacje graficzne, nie odrywając wzroku od ekranu.

Na najniższym poziomie oprogramowania graficznego znajduje się biblioteka *Pixrect*, podstawa oprogramowania graficznego. *Pixrect* jest oprogramowany w języku obiektowym, wykorzystującym zalety urządzeń ekranowych. Obiekty są definiowane przez współrzędne (całkowite) punktów ekranu, a operatory umożliwiają działanie na wektorach, obszarach, tekstach, strukturach powierzchni (teksturach), a także na kolorowanie powierzchni.

Korzystanie z biblioteki *Pixrect* jest efektywne, ale nie pozwala na budowanie obiektów, które mogłyby być przenoszone na inne komputery. Skłoniło to producenta do wprowadzenia trzech dodatkowych standardów graficznych:

- *SunCore* jest to system oparty na pierwszej amerykańskiej normie graficznej *ACM Core*. Pakiet ten zawiera zbiór elementów podstawowych, służących do opisu obiektów dwuwymiarowych i trójwymiarowych we współrzędnych rzeczywistych, oraz funkcje służące do wyświetlania tych obiektów z różnej perspektywy i w różnym oświetleniu. Ponadto *SunCore* umożliwia dzielenie obiektów na segmenty (zbiory elementów podstawowych). Segmenty mogą być tworzone, usuwane, wybierane i rozjaśniane przy użyciu myszy. Zbiór elementów podstawowych zawiera wektory, wielokąty, punkty, teksty i maski. Każdy z takich obiektów może mieć nadane atrybuty koloru i tekstury, a także może być wyświetlony bez uwidoczniania powierzchni przesłanianych. Obiekty mogą być pocieniowane w sposób płynny, co łagodzi wrażenie ostrych przejść pomiędzy ściankami.
- *SunCGI* (ang. *computer graphics interface)* jest zgodny z normą opracowaną przez ANSI. Zawiera bardzo bogatą bibliotekę elementów podstawowych i struktur. Jest w pełni zintegrowany z systemem okien i może współpracować z programami napisanymi w języku C. W odróżnieniu od *SunCore*, umożliwia korzystanie z funkcji najniższego poziomu i dostęp do urządzeń graficznych. Nie dopuszcza natomiast dzielenia obiektów, co przyspiesza ich wyświetlanie, ale nie daje możliwości automatycznej regeneracji obrazu. Programy napisane przy użyciu *SunCGI* są zazwyczaj mniejsze niż analogiczne programy dla *SunCore*.
- *SunGKS* (ang. *graphical kernel system)* jest zgodny z normą ISO. Wersja *SunGKS* zawiera pełny poziom *2C* (dwuwymiarowy). *SunGKS* jest podobny do *SunCore* – główna różnica polega na znacznie większej niż w *SunCore* precyzji normy. Dzięki temu jest on bardziej niezależny od sprzętu. Zawiera ponadto opis plików transferowych *GKSM*, co znacznie rozszerza zasięg stosowania produktów graficznych wywodzących się z *GKS*.

Wszystkie wymienione systemy graficzne można wykonywać przez *SunWindows* w wielu oknach, które mogą być traktowane jako wirtual-

ne urządzenia zewnętrzne. Obiekty tworzone przy użyciu *SunCore, SunCGI i SunGKS* mogą być oglądane w oknach, a przy zmianie parametrów okien obiekty automatycznie dostosowują się do tych zmian.

Dzięki szybkim procesom bipolarnym i szybkim koprocesorom arytmetycznym stało się możliwe przetwarzanie potokowe w grafice. Firma Sun na życzenie dostarcza koprocesor graficzny, działający w trybie potokowego przetwarzania danych, i graficzny bufor; urządzenia te przyspieszają przetwarzanie od 2 do 250 razy, zależnie od operacji. Koprocesor graficzny umożliwia grafikę okienkową, wieloprocesowość i jednoczesność operacji w wielu oknach. Bufor graficzny jest dodawany przy modelowaniu obiektów trójwymiarowych, w systemach *CAD* i przy kolorowaniu obiektów. Dzięki graficznemu koprocesorowi i buforowi możliwa stała się okienkowa grafika trójwymiarowa w tanich stacjach roboczych. Koprocesor graficzny umożliwia kreślenie 4000 dwuwymiarowych wielokątów na sekundę. Operacje na punktach są wykonywane około ośmiu razy szybciej, a na matrycach – przy kolorowaniu – trzy do czterech razy szybciej niż w stacjach bez koprocesora.

Programowanie grafiki, tak jak każda inna działalność związana z programowaniem, może być prowadzone w lokalnej sieci komputerowej. Sieciowy system plików może być używany do konstruowania graficznych baz danych, bez konieczności wielokrotnego kopiowania graficznych zbiorów danych (na ogół bardzo dużych). Dzięki temu staje się możliwe używanie tych samych plików transferowych (np. GKSM w SunGKS) przez wszystkich użytkowników sieci, a ponadto zostaje wyeliminowany problem zgodności wielu kopii tych samych plików. Drugą istotną zaletą pracy w sieci jest możliwość wykorzystywania komputera usługodawcy (ang. *server)* do pracy z bazą danych, dzięki czemu stacja robocza jest mniej obciążona i służy tylko do pracy z obiektami graficznymi. Powyższa zaleta jest możliwa dzięki procedurom *RPC* uruchomianym w sieciowym systemie plików *NFS*.

MICHAŁ HORNOWSKI
Instytut Technologii Elektronowej
NPCP CEMI
Warszawa

Metoda instalowania bardzo dużych programów

Podstawowym problemem po zakupie programu jest jego zainstalowanie na posiadanym sprzęcie i w środowisku systemu operacyjnego działającego na tym sprzęcie. Szczególnie skomplikowane jest instalowanie bardzo dużych programów – mimo zapisania ich w języku wysokiego poziomu. W artykule przedstawiono metodę instalowania dużych programów, wypracowaną w Instytucie Technologii Elektronowej. Wyróżniono w niej kilka etapów, których kolejna realizacja ułatwia całość pracy.

PRZENIESIENIE TEKSTU ŹRÓDŁOWEGO PROGRAMU I PLIKÓW POMOCNICZYCH

Na tym etapie zwykle występują trudności wynikające z odmiennego systemu zapisu treści programu na nośniku. Zapisy mogą różnić się kodami znaków, długością rekordów i bloków, gęstością zapisu, liczbą ścieżek, bitami kontrolnymi, nagłówkami plików itp. Często występują też przekłamania przy próbach odczytu. Są one związane z niedoskonałością sprzętu (np. złe ustawienie głowic odczytujących) lub złym stanem nośnika (np. załamanie taśmy).

Odczytanie nietypowego formatu wymaga specjalnych programów. Programy te zwykle nie uwzględniają wszystkich możliwości kontroli poprawności odczytu. Z tego powodu nie wszystkie błędy odczytu są sygnalizowane i korygowane.

Część przekłamań można zlokalizować czytając zapis dwukrotnie i porównując obie wczytane wersje. Należy też sprawdzić liczbę linii tekstu w celu wykrycia ewentualnego pominięcia fragmentów. Porównywanie tekstów można zautomatyzować. W ITE opracowano w tym celu specjalny interakcyjny program. Błędy wynikające z przekłamań należy poprawić przy użyciu edytora tekstów. Wątpliwości można wyjaśnić korzystając z wydruków, zwykle dołączanych przez dostawcę do programów zapisanych na taśmie.

Po odczytaniu całego materiału otrzymanego z innego ośrodka należy przygotować dobrze zabezpieczone kopie wszystkich wczytanych plików (np. na taśmie magnetycznej). Kopie te mogą okazać się przydatne wówczas, gdy z jakiegoś powodu podczas dalszych etapów uruchamiania nastąpi zniszczenie istotnych fragmentów oprogramowania.

Numeracja linii i podział programu

Większość języków programowania ma mechanizmy umieszczania komentarzy (a więc również numerów) w liniach tekstu źródłowego programu. Linie programu powinny być ponumerowane. Jeżeli nie zrobił tego autor, to trzeba ponumerować je samemu. Dobrze jest ustalić taki sposób numerowania, aby było możliwe oznaczanie poprawek. W przyszłości umożliwi to szybkie lokalizowanie dokonanych zmian.

Autor programu zwykle tworzy go w częściach, zawierających pewne fragmenty logicznie powiązane ze sobą. Znajduje to odbicie w wersji programu przeznaczonej do rozpowszechniania. Czasami moduły są ustawione alfabetycznie lub w innym porządku. Ze względu na znaczne oszczędności czasu procesora, w dalszych etapach uruchamiania należy tekst programu podzielić na części po około 1000 do 2000 linii. Najbardziej naturalny jest podział odpowiadający strukturze programu, np. część I – obsługa wejścia i wyjścia, część II – analizator składni, część III – rozwiązywanie równań, część IV – dostęp do bazy danych.

Jeżeli jednak struktura programu nie jest znana, to można podzielić go na części zbliżone wielkością, na przykład po około 1500 linii. Liczba części może być ograniczona możliwościami konsolidatora. Trzeba pamiętać o tym, że w każdej części musi znaleźć się całkowita liczba modułów, oraz że musi być zapewniona komunikacja między modułami z różnych części.

Kompilacja i konsolidacja

Kompilacja każdej części odbywa się identycznie, a więc można opracować odpowiedni zbiór poleceń, dla których parametrem jest numer lub nazwa części. Należy stosować kompilator o dużych możliwościach diagnostycznych. Po pierwszej próbie kompilacji każdej z części programu, należy przeanalizować wykryte przez kompilator błędy. Niektóre z nich powtarzają się we wszystkich częściach (lub w większości), inne są lokalne. Pierwszą grupę błędów eliminuje się przez redagowanie z użyciem pliku zawierającego odpowiednie dyrektywy edytora. Pozostałe zaś usuwa się w sposób tradycyjny, wprowadzając kolejno poprawki. Konieczność wprowadzenia poprawek może wynikać z różnic w sprzęcie (np. inna długość słowa), z użycia innego kompilatora (np. różnice w słowach kluczowych), z błędów kodowania nie zauważonych przez autora (np. *I0* zamiast *IO*) itd. Wszystkie zmiany należy zaznaczyć w polu numeracji z rozróżnieniem charakteru zmian: istotniejsze zmiany należy komentować w tekście programu. Po wprowadzeniu poprawek każdą z części ponownie poddaje się próbie kompilacji.

Zmiany wprowadza się aż do otrzymania komunikatu o bezbłędnej kompilacji. Należy również dążyć do wyeliminowania wszystkich ostrzeżeń (ang. *warnings*). Ostatnie pliki wynikowe kompilacji – *object modules (DEC), semicompiled segments (ICL)* – trzeba zachować. Dla zapewnienia ochrony plików należy mieć zawsze zachowaną poprzednią wersję źródłową kompilowanego fragmentu. Zbędne pliki należy usuwać.

Po udanej kompilacji wszystkich części można podjąć próbę konsolidacji programu. Pierwsza próba dostarcza informacji o nieprawidłowych odwołaniach międzymodułowych oraz o wielkości programu. Należy poprawić odpowiednie części programu i ewentualnie dołączyć brakujące moduły. Brakujących modułów szuka się w różnych bibliotekach systemowych, w dokumentacji danego programu, w innych programach tych samych autorów, ewentualnie trzeba je napisać od nowa na podstawie analizy spełnianych funkcji. Może też zajść konieczność dopisania podprogramów w kodzie maszynowym. Po przeprowadzeniu kolejnych kompilacji ponawia się próby konsolidacji.

Nakładkowanie, skracanie i testowanie

Jeżeli nie dysponujemy komputerem z pamięcią wirtualną, to zwykle konieczne jest opracowanie systemu nakładek. Podstawą tego opracowania jest drzewo odwołań od podprogramów. Informacja o strukturze drzewa odwołań powinna znajdować się w dokumentacji programu. Jeżeli jej tam nie ma, to należy analizować kolejne podprogramy i zapisywać odwołania do dalszych podprogramów. Niektóre kompilatory mają opcję, umożliwiającą wydrukowanie wszystkich odwołań w danym segmencie: na tej podstawie można zbudować drzewo odwołań. Sposób opisu nakładek jest związany z właściwościami konkretnej instalacji.

Może się zdarzyć, że program – mimo zastosowania nakładek – jest zbyt duży jak na możliwości naszego komputera. Należy go wtedy

skrócić. Najprostsze sposoby skracania polegają na zmniejszaniu rozmiarów tablic, zmniejszaniu liczby słów pamięci zajmowanych przez różne typy zmiennych oraz na wyłączaniu z konsolidacji wybranych modułów. Pierwszy sposób zwykle wprowadza ograniczenia ilościowe w danych, drugi – powoduje zmniejszenie dokładności obliczeń, trzeci natomiast wprowadza ograniczenia jakościowe w danych. Każda z tych metod może powodować złe funkcjonowanie programu. W celu skrócenia można też zastosować kompilator optymalizujący, należy jednak pamiętać, że jego zastosowanie zwykle utrudnia uruchamianie programu.

Po uzyskaniu prawidłowo skonsolidowanego programu można rozpocząć jego testowanie. Poprawna dokumentacja powinna zawierać pewną liczbę testów. Jeżeli tak nie jest, to testy trzeba opracować samemu. Zwykle powstaje przy tym problem prawidłowego wprowadzania danych i wyprowadzania wyników. Później testuje się kolejne główne moduły. W trakcie testowania trzeba intensywnie korzystać z narzędzi do diagnostyki, zarówno wbudowanych w program przez autorów (np. wydruki kontrolne), jak i związanych z możliwościami systemu operacyjnego (np. użycie programu uruchomieniowego). Czasem zachodzi potrzeba opracowania specjalnego programu do przetestowania pewnych fragmentów uruchamianego programu. Umożliwia to dokładne testowanie wybranych modułów, które wydziela się z uruchamianego programu i włącza do programu diagnostycznego. Bezwzględnie konieczne jest prowadzenie starannej dokumentacji poprawek wprowadzanych do wersji źródłowej.

Eksploatacja

Po zakończeniu testowania, poprawieniu błędów i zaznaczeniu wszystkich wprowadzonych zmian, program przygotowuje się do próbnej eksploatacji. Należy dążyć do tego, aby przyszły użytkownik mógł rozpocząć eksploatację programu mając jak najmniej informacji o samym programie i o systemie, w jakim program ten funkcjonuje. W tym celu należy:
– wyznaczyć w zasobach pamięci systemu obszar dla użytkowników programu,
– opracować pliki poleceń systemowych, uruchamiających dany program na odpowiednie hasło,
– opracować instrukcję przygotowania danych oraz instrukcję obsługi programu,
– przygotować kilka sprawdzonych przykładów,
– ustalić osoby, które będą udzielały użytkownikom konsultacji w dziedzinie funkcjonowania programu.

Konieczna jest ochrona istotnych plików przed nieumyślnym zniszczeniem (np. na taśmie magnetycznej). Należy zachować ostatnie wersje źródłowe i półskompilowane wszystkich części programu, wersję binarną programu, stałe pliki danych oraz pliki poleceń do kompilacji, konsolidacji i uruchamiania. Zbędne pliki można usunąć.

Po okresie próbnej eksploatacji, podczas której może zajść konieczność poprawiania niektórych fragmentów, przygotowuje się normalną użytkową wersję programu. W wersji tej wyłącza się znaczną część mechanizmów diagnostycznych, przywraca się wszystkie poprzednio okrojone możliwości programu, optymalizuje (w miarę możliwości) wielkość i szybkość działania programu, wprowadza udogodnienia zaproponowane przez użytkowników podczas próbnej eksploatacji. Kopie wszystkich istotnych plików należy zredagować tak, jakby miały być przeniesione na inną maszynę tego samego typu i zabezpieczyć przed zniszczeniem. Oddanie programu do normalnej eksploatacji wymaga jeszcze pewnych ustaleń organizacyjnych (np. ustalenia godzin użytkowania przez różnych użytkowników).

* * *

Według opisanej metody do listopada 1985 uruchomiono na dwóch komputerach kilkanaście dużych programów napisanych w Fortranie.

Metodą tradycyjną (tj. bez podziału na części) podjęto próbę uruchomienia pięciu dużych programów. W jednym wypadku próba powiodła się, w jednym nie powiodła się, w dwóch zaszła konieczność podzielenia programu na znacznie różniące się wielkością części, a w jednym wypadku prac zaniechano ze względu na duże niedogodności (uruchamianie dokończono metodą zaproponowaną w artykule).

Opisaną metodę warto stosować dla programów dłuższych niż 3000 linii kodu źródłowego, o ile program nie działa poprawnie po pierwszej kompilacji i konsolidacji.

W dalszych pracach przewiduje się porównanie opisanych w artykule doświadczeń z amerykańską normą wojskową dotyczącą dokumentacji oprogramowania.

STANISŁAWA OSSOWSKA
Katedra Organizacji Przetwarzania Danych
Uniwersytet Gdański

Dobór metody projektowania systemów informatycznych zarządzania

Jednym z istotnych problemów warunkujących skuteczność zastosowania informatyki do przetwarzania danych w przedsiębiorstwach jest dobór odpowiednich metod projektowania systemów informatycznych zarządzania. Biorąc pod uwagę fakt tradycyjnego zarządzania (sięgający jeszcze czasów Kartezjusza), sądzę że badania teoretyczne w zakresie doboru metod, mogące zmniejszyć częstotliwość popełnianych błędów praktycznych, są ze wszech miar uzasadnione.

PROJEKTOWANIE JAKO ROZWIĄZYWANIE PROBLEMÓW ORGANIZACYJNYCH

Punktem wyjścia do sformułowania zasady doboru metody projektowania jest przyjęcie założenia, że **projektowanie systemów informatycznych zarządzania (SIZ) może i powinno być traktowane jako rozwiązywanie problemów organizacyjnych. Stwierdzenie to wynika między innymi stąd, że projektowanie SIZ sprowadza się do rozwiązywania określonych problemów, występujących w ramach istniejącej organizacji, a zadania projektowanego SIZ powinny być podporządkowane celom tej organizacji.**

Rozwiązanie problemu organizacyjnego polega na budowie nowego, pożądanego systemu, na którą składają się dwa podstawowe etapy:
1) poszukiwanie (projektowanie) w danych warunkach rozwiązania optymalnego, czyli tworzenie projektu, a najpierw koncepcji projektowanego systemu,
2) realizacja projektu, czyli przygotowanie warunków techniczno-organizacyjnych i psychiczno-społecznych przed przystąpieniem do stopniowego wdrażania projektu, wdrażanie, a następnie kontrola eksploatacji nowego systemu.

Projektowanie systemów bywa często utożsamiane z ich budową, o czym świadczą poszczególne etapy projektowania obejmujące urzeczywistnianie projektu, wyróżniane przez wielu autorów. Takie określenie nie wydaje się słuszne, ponieważ projektowanie nie obejmuje realizacji przedsięwzięcia. Projektowanie sprawdza się ostatecznie dopiero w procesie funkcjonowania zrealizowanego systemu, co pozwala na zweryfikowanie projektu i dokonanie jego faktycznej oceny.

Biorąc pod uwagę różne kryteria można wyróżnić kilka rodzajów problemów organizacyjnych (por. rysunek). Przykładowo, ze względu na stopień złożoności problemu wyodrębnia się przedsięwzięcia:
- stosunkowo proste, możliwe do zrealizowania przy wykorzystaniu nieskomplikowanych i powszechnych technik badawczych,
- złożone, wymagające pracy specjalistów-konsultantów,
- bardzo skomplikowane, na ogół dotyczące tworzenia nowoczesnych systemów organizacyjnych, gdzie obok typowych technik organizatorskich winny być stosowane inne techniki, łącznie z przetwarzaniem danych.

Innego podziału problemów można dokonać ze względu na stan, w którym znajduje się przedmiot organizowania.

Według takich samych kryteriów dokonuje się podziału problemów informatycznych, których rozwiązywanie wymaga doboru odpowiednich metod.

Schemat poszukiwania rozwiązania przy zastosowaniu metody diagnostycznej i prognostycznej

Projektowanie analityczne (diagnostyczne)	Projektowanie syntetyczne (prognostyczne)
Sformułowanie problemu	Określenie funkcji (celu) projektowanego systemu
Zbieranie szczegółowych informacji o funkcjonowaniu istniejącego dotychczas systemu	Sformułowanie założeń systemu idealnego
Analiza systemu istniejącego	Zbieranie informacji o warunkach, w których ma funkcjonować system projektowany
Krytyka stanu istniejącego i tworzenie alternatywnych koncepcji systemu	Zaprojektowanie możliwych do realizacji wariantów
Ocena różnych rozwiązań i wybór koncepcji optymalnej	Wybór rozwiązania optymalnego

CHARAKTERYSTYKA METOD PROJEKTOWANIA

Poniżej przedstawiono dwie metody projektowania systemów wskazując na ich różnice w świetle procesu tworzenia koncepcji systemu oraz uzyskiwanych cech projektowanego systemu.

Projektowanie według tworzenia koncepcji systemu

Zasadniczo wyróżnić można dwa odmienne sposoby podejścia do rozwiązywania problemów organizacyjnych – podejście analityczne oraz podejście idealne. Wybór podejścia przesądza o metodzie projektowania. Tak więc, w zależności od punktu wyjścia przy poszukiwaniu rozwiązania, wyróżnia się:
- metodę diagnostyczną (analityczną, tradycyjną),
- metodę prognostyczną (syntetyczną, wzorca idealnego).

Wymienione metody projektowania różnią się pomiędzy sobą nie tylko innym punktem wyjścia, przyjmowanym dla rozwiązywania problemów, lecz także odmiennym sposobem tworzenia koncepcji systemu.

Punktem wyjścia dla koncepcji tworzonej metodą analityczną jest stan dotychczasowy. Rozwiązanie problemu występującego w funkcjonującym już obiekcie ma na celu poprawienie sprawności jego działania. Rozpoczynając od stanu dotychczasowego, w metodzie analitycznej mówi się o projektowaniu wstępującym. Z kolei w metodzie prognostycznej za punkt wyjścia przyjmuje się określenie celów (funkcji), które realizować ma nowy, projektowany system. Założone cele są podstawą do wypracowania koncepcji systemu idealnego, nie obciążonego żadnymi ograniczeniami. Taki sposób rozwiązywania problemu określany jest jako projektowanie zstępujące.

Sposób wypracowywania koncepcji jest ściśle związany z punktem wyjścia dla poszukiwanego rozwiązania. W metodzie analitycznej, bezpośrednio po sformułowaniu problemu, przystępuje się do gromadzenia szczegółowych informacji o poszczególnych elementach składowych istniejącego już systemu. Zebrane informacje pozwalają na przeprowadzenie krytycznej analizy stanu istniejącego, mającej na celu wykrycie przyczyn niesprawności istniejącego systemu. Pozwala to na poszukiwanie sposobów usprawnienia i przekształcenia systemu do nowej, doskonalszej postaci. Rozwiązywanie wypracowywane jest na drodze analiza-synteza-ocena i dotyczy problemów o charakterze dewiacyjnym bądź optymalizacyjnym.

W metodzie syntetycznej, bezpośrednio po określeniu celów projektowanego systemu, tworzy się koncepcję systemu idealnego. System idealny stanowi podstawę do wypracowania systemu możliwego do

zrealizowania zgodnie z istniejącymi warunkami. Informacje o warunkach, w których ma działać system idealny, są zbierane w dalszym etapie i dotyczą jedynie możliwości realizacji systemu. Przy wykorzystaniu tej metody projektowanie odbywa się na drodze synteza-analiza-ocena.

Syntetyczny schemat postępowania stosowany przy projektowaniu systemów obydwiema metodami przedstawiono w tabeli. Opisano w niej czynności dotyczące tylko pierwszego etapu budowy systemu, czyli etapu projektowania. Etap realizacji systemów przebiega podobnie, niezależnie od stosowanej metody projektowania.

Projektowanie systemu według uzyskiwanych cech

Różnice między systemami projektowanymi przy zastosowaniu metody diagnostycznej i prognostycznej występują w postaci poziomu jakości projektu oraz pracochłonności, czasochłonności i kosztów prowadzonych prac projektowych.

Poziom jakości projektu, czyli poziom rozwiązania systemu, wypracowanego na podstawie koncepcji stanu idealnego jest wyższy od poziomu rozwiązania systemu otrzymanego metodą diagnostyczną [1, 2]. Zróżnicowanie poziomu jakości systemów otrzymywanych przez zastosowanie dwu odmiennych metod projektowania wynika z wielu przyczyn.

W podejściu analitycznym uwaga zespołu projektującego jest skoncentrowana bardziej na składnikach systemu, niż na całości, co w konsekwencji może doprowadzić do optymalizacji jedynie pewnych elementów systemu. Wiadomo, że nie zawsze optymalizacja rozwiązań cząstkowych jest jednoznaczna z optymalizacją rozwiązania kompleksowego. Ponadto, przyjęcie za punkt wyjścia systemu istniejącego może stwarzać bariery utrudniające budowę nowego systemu. W procesie projektowania systemu analiza ogranicza wyobraźnię zespołu projektującego, przez co rozwiązanie problemu jest zwykle modyfikacją rozwiązań już istniejących i nie uwzględnia przyszłościowych warunków techniczno-ekonomicznych. W procesie wdrażania systemu analiza może być przyczyną negatywnych reakcji ludzi odpowiedzialnych za dotychczasowy stan, który był przedmiotem krytycznej analizy.

Podejście syntetyczne pozwala na skoncentrowanie uwagi zespołu na istocie problemu jako całości, bez niebezpieczeństwa przytłoczenia szczegółami. Metoda prognostyczna stwarza warunki do wyzwolenia aktywności, daje szanse proponowania szerokiego repertuaru śmiałych i oryginalnych rozwiązań, wśród których łatwiej jest znaleźć rozwiązanie optymalne dla określonych warunków. Przy stosowaniu tej metody cała uwaga powinna być skupiona na poszukiwaniu rozwiązań idealnych, nie obciążonych istniejącym stanem rzeczy. Zapobiega to negatywnym reakcjom ludzi, którzy – zamiast krytykować dotychczasowe rozwiązania – od razu przystępują do poszukiwania nowych rozwiązań, łatwiej przyjmują nowe rozwiązania i łatwiej je wdrażają.

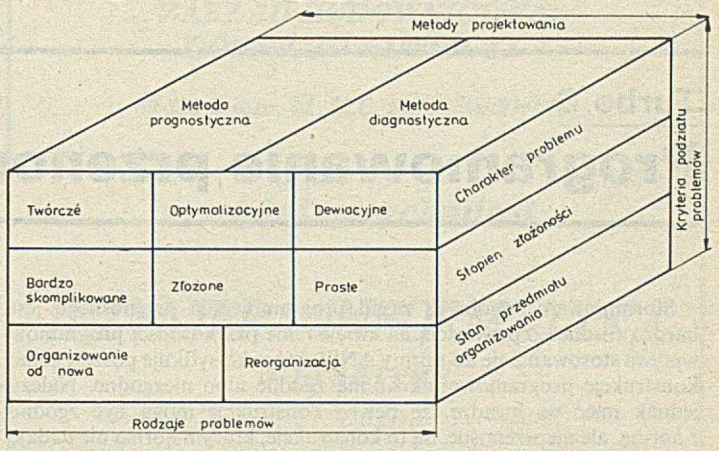
Czasochłonność i koszty prac projektowych są zróżnicowane przez wymaganą liczbę prac, związanych z gromadzeniem i analizą informacji dotyczących funkcjonowania systemu dotychczasowego.

Projektowanie przyjmujące za punkt wyjścia koncepcję rozwiązania idealnego wymaga zbierania znacznie mniejszej ilości informacji, gdyż dotyczą one nie sposobu funkcjonowania dotychczasowego systemu, lecz warunków, w jakich projektowany system ma działać. Przyjmuje się, że metoda analityczna wymaga zaangażowania większych środków, a co za tym idzie – ponoszenia większych nakładów. Ponadto wydłuża okres projektowania, dając jednocześnie gorsze jakościowo rozwiązania.

Rozważając problem czasochłonności projektowania należy pamiętać o dynamicznych zmianach zarówno w technice, jak i organizacji, skracających okresy żywotności istniejących systemów, na przykład organizacyjnego, zarządzania, technicznego, informacyjnego itp. Taki stan rzeczy wymaga maksymalnego skracania okresu prac projektowo-wdrożeniowych przez dobór odpowiednich metod i technik rozwiązywania problemów, właściwą organizację prowadzonych prac oraz ich intensywność.

DOBÓR METODY PROJEKTOWANIA

Zasadą doboru metody projektowania stosownie do rodzaju problemu przedstawiono na rysunku. Należałoby rozważyć, do jakich kategorii problemów można zaklasyfikować projektowanie systemów informatycznych, jeśli wiadomo, że sytuacja poszczególnych jednostek organizacyjnych decydujących się na podjęcie prac projektowania systemów przetwarzania danych jest bardzo zróżnicowana.



Zasada doboru metody projektowania w zależności od rodzaju problemu

W pierwszym wypadku może to być organizacja, w której już funkcjonuje system informatyczny. Ma on poprawnie skonstruowaną generalną koncepcję, charakteryzującą się kompleksowością rozwiązania problemów informatycznych oraz uwzględniającą zasadę współzależności pomiędzy stopniem kompleksowości a stopniem zintegrowania systemu informatycznego zarządzania [3]. W myśl tej zasady wraz ze wzrostem stopnia kompleksowości systemu powinien wzrastać stopień jego zintegrowania, rozpatrywany w płaszczyźnie rzeczowej, czynnościowej i atrybutowej. W takiej sytuacji występujące problemy informatyczne mają zwykle charakter problemów dewiacyjnych bądź optymalizacyjnych i są w niewielkim stopniu złożone. System informatyczny wymaga wówczas jedynie reorganizacji, czyli modyfikacji niektórych jego elementów. Dla rozwiązania tego typu problemów sensowne jest stosowanie metody diagnostycznej.

Innego rodzaju problemy informatyczne wystąpią wówczas, gdy określona organizacja stosuje sprzęt komputerowy po raz pierwszy. W wypadku kompleksowego ujęcia sprowadza się to do rozwiązywania problemów bardzo złożonych, o charakterze twórczym, a niekiedy również optymalizacyjnym. Podobnie przedstawia się sytuacja, gdy w ramach istniejącej organizacji stosowano już wprawdzie sprzęt komputerowy ale tylko dla pewnych wycinków działalności, oraz gdy rozwiązania miały charakter wycinkowy i nie uwzględniały całokształtu działalności informatycznej, możliwej do zrealizowania przy dostępnym wyposażeniu technicznym. Właściwe rozwiązanie scharakteryzowanej grupy problemowej wymagałoby utworzenia projektu kompleksowego, a więc zaprojektowania systemu informatycznego od nowa, co wymaga podejścia idealnego. W takich wypadkach powinna być stosowana metoda prognostyczna.

Z powyższych rozważań wynika, że **do projektowania SIZ mogą być stosowane obydwie metody projektowania, zarówno metoda diagnostyczna, jak i prognostyczna.** Dobór metody powinien być warunkowany stopniem nowości, a tym samym trudności problemu, jego złożoności i rozległości w konkretnych organizacjach. Rozstrzygnięcie o zakwalifikowaniu projektowania SIZ do jednej z dwóch modelowych grup problemów może nastąpić po wyspecyfikowaniu i ocenie czynników składających się na realne warunki tej organizacji.

Nasuwa się tu kolejny problem praktyczny, jakim jest umiejętność stosowania metody prognostycznej przez zespoły projektujące SIZ. Myślę, że jeśli nawet dzisiejsza ocena tych umiejętności byłaby niezadowalająca, uznanie zasadności stosowania metody prognostycznej jest zaczątkiem do zmiany istniejącego stanu rzeczy.

LITERATURA

[1] Kolbusz E., Kram E.: Wdrażanie systemów informatycznych w przedsiębiorstwie przemysłowym. PWE, Warszawa, 1977

[2] Niedzielska E. (Red.): Informatyka. Wstęp do informatyki. Technologia przetwarzania danych. PWE, Warszawa, 1983

[3] Ossowska S.: Cechy modeli systemów informatycznych zarządzania i ich wpływ na jakość tworzonych systemów. Materiały konferencji „Rachunkowość i informatyka w zarządzaniu przedsiębiorstwem". Łódź-Spała, 1984.

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Turbo C

Programowanie przenośne

Sformułowanie ogólnych zasad programowania przenośnego jest bardzo trudne. Z pewnością na zwiększenie przenośności programów wpływa stosowanie się do normy ANSI, która klasyfikuje poszczególne konstrukcje programowe jako z nią zgodne albo niezgodne. Należy jednak mieć na uwadze, że pewne konstrukcje mogą być zgodne z normą, ale nieprzenośne. Są to konstrukcje, którym norma nie nadaje interpretacji, bądź konstrukcje, w stosunku do których norma formułuje wymaganie, aby interpretacja została podana w opisie implementacji.

Zachowanie nieokreślone

Konstrukcją wywołującą zachowanie nieokreślone jest każda konstrukcja poprawna, dla której nie określono interpretacji.

Przykłady

Źródłem zachowań nieokreślonych jest wykorzystanie:
- sposobu i momentu wykonania statycznych inicjalizacji,
- skutku wyprowadzenia znaku widocznego, gdy kursor znajduje się na końcu wiersza,
- skutku wyprowadzenia znaku '\b', gdy kursor znajduje się na początku wiersza,
- skutku wyprowadzenia znaku '\t', gdy kursor znajduje się na ostatniej lub poza ostatnią (poziomą) pozycją tabulacyjną,
- skutku wyprowadzenia znaku '\v', gdy kursor znajduje się na ostatniej albo poniżej ostatniej (pionowej) pozycji tabulacyjnej,
- kolejności opracowania wyrażeń,
- kolejności wystąpienia skutków ubocznych,
- kolejności opracowania argumentów funkcji,
- sposobu rozmieszczenia w pamięci operacyjnej parametrów funkcji,
- sposobu rozmieszczenia obszarów przydzielonych przez funkcje do zarządzania pamięcią operacyjną,
- sposobu rozmieszczenia „równych" elementów tablicy posortowanej za pomocą funkcji *sort*,
- sposobu zakodowania pory czasu dziennego udostępnionego przez funkcję *time*.

Zachowanie niezdefiniowane

Konstrukcją wywołującą zachowanie niezdefiniowane jest każda konstrukcja błędna.

Przykłady

Błędem programowania jest:
- przypisanie danej – elementowi ciągu zmiennych reprezentowanych w programie przez literał napisowy,
- posłużenie się nieidentycznymi deklaracjami identyfikatora (tego samego obiektu) występującego w różnych modułach źródłowych,
- posłużenie się literałem zawierającym napis \c, w którym *c* jest małą literą różną od *x*, *a*, *b*, *f*, *n*, *r*, *t*, *v*,
- utworzenie danej, która nie może być reprezentowana w wymaganym obszarze pamięci,
- próba ponownej definicji zmiennej ustalonej,
- próba ponownej definicji zmiennej nietrwałej za pomocą wskazania przypisanego takiej zmiennej, która nie służy do wskazywania zmiennych nietrwałych,
- wykonanie niepoprawnej operacji arytmetycznej, takiej jak dzielenie przez 0, albo wykonanie operacji powodującej powstanie nadmiaru,
- wywołanie funkcji, dla której nie było deklaracji prototypowej, z niewłaściwą liczbą argumentów lub z argumentami, które (po dokonaniu niejawnych konwersji) nie są zgodne z parametrami funkcji,
- wywołanie funkcji, która akceptuje wywołania ze zmienną liczbą argumentów, bez użycia deklaracji prototypowej zawierającej symbol ... (trzy kropki),
- odwołanie się do nie istniejącego elementu tablicy,
- użycie operatora wyłuskania lub indeksowania w odniesieniu do wskazania pustego,
- odwołanie się do zmiennej automatycznej, która przestała istnieć,
- wykonanie operacji arytmetycznej, której jednym z argumentów jest wskazanie zmiennej, która nie jest elementem tablicy,
- wyznaczenie różnicy wskazań zmiennych nie należących do tej samej tablicy,
- porównanie wskazań zmiennych nie należących do tego samego agregatu (struktury albo tablicy),
- posłużenie się daną o wartości nieokreślonej,
- wywołanie funkcji, zdefiniowanej w zasięgu deklaracji prototypowej, poza zasięgiem równoważnego prototypu,
- użycie w preprocesorze operatora ##, które nie zostanie zakończone wygenerowaniem identyfikatora,
- użycie w preprocesorze dyrektywy #*undef* w celu uzyskania dostępu do nie istniejącej funkcji bibliotecznej,
- wywołanie funkcji z argumentem spoza dziedziny tej funkcji,
- w okresie między wywołaniami funkcji *setjmp* i *longjmp*, przypisanie zmiennej automatycznej nie będącej nietrwałą, danej o nowej wartości,
- wywołanie funkcji *longjmp* z funkcji sygnałowej wywołanej z innej funkcji sygnałowej,
- wywołanie funkcji, która może być wywoływana ze zmienną liczbą argumentów, z takim argumentem, którego typ nie jest zgodny z typem określonym w makrowywołaniu *va_arg*,
- zakończenie wykonywania funkcji, w której wywołano makrodefinicję *va_start*, ale nie wykonano makrodefinicji *va_end*,
- wywołanie funkcji *fprintf* albo *fscanf* (lub im podobnej) z argumentem, którego typ jest niezgodny z wzorcem konwersji,
- wywołanie funkcji *fprintf* albo *fscanf* (lub im podobnej) z wzorcem konwersji zawierającym znak, któremu nie nadano interpretacji,
- wywołanie funkcji *fprintf* (lub jej podobnej) z argumentem, który reprezentuje strukturę albo tablicę,
- cofnięcie znaku do pliku, jeśli w pliku znajduje się cofnięty uprzednio znak,
- wywołanie funkcji z argumentem spoza jej dziedziny,
- odwołanie się do zmiennej znajdującej się w zwolnionym obszarze pamięci.

Zachowanie zależne od implementacji

Konstrukcja zależna od implementacji jest elementem programu poprawnego, ale nie jest przenośna.

Przykłady

Zależne od implementacji jest:

W kategorii **Środowisko**
- określenie semantyki argumentów funkcji *main*.

W kategorii **Identyfikatory**
- określenie liczby istotnych znaków identyfikatora (powyżej 31

w wypadku identyfikatorów lokalnych w module i powyżej 6 w wypadku identyfikatorów międzymodułowych),
- określenie, czy w identyfikatorze międzymodułowym litery duże i małe uchodzą za jednakowe.

W kategorii **Dane znakowe**
- wyszczególnienie pełnego alfabetu znaków dostępnych podczas wykonywania programu,
- wyszczególnienie pełnego alfabetu znaków dostępnych podczas kompilowania programu,
- uporządkowanie znaków w danej typu (*int*),
- określenie liczby bitów niezbędnych do reprezentowania znaku,
- określenie odwzorowania znaków dostępnych podczas kompilowania programu, zawartych w literałach znakowych i napisowych, na znaki dostępne podczas wykonywania programu,
- wystąpienie po ukośniku zawartym w literale znakowym lub napisowym takiego znaku, któremu nie nadano interpretacji,
- określenie wartości danej reprezentowanej przez literał znakowy zawierający więcej niż jeden znak,
- określenie czy dana nazwa typu *char* jest równoważna nazwie *signed* czy nazwie *unsigned char*.

W kategorii **Dane całkowite**
- określenie sposobu reprezentowania i zakresu wartości danych,
- określenie skutków konwersji danej typu (*int*) na daną typu (*short int*) lub daną typu (*signed char*), jeśli miałoby to spowodować zmianę wartości danej,
- określenie skutków wykonania operacji bitowych,
- określenie znaku reszty w dzieleniu całkowitym,
- określenie skutku użycia operatorów przesunięcia, jeśli prawy argument operacji reprezentuje daną o wartości ujemnej, lub daną o wartości równej lub większej od liczby bitów danej całkowitej,
- określenie czy przesuwanie w prawo danej typu (*signed int*) jest arytmetyczne czy logiczne.

W kategorii **Dane zmiennopozycyjne**
- określenie sposobu reprezentowania i zakresu wartości danych,
- określenie sposobu zaokrąglania w wypadku konwersji danych typu (*double*) na dane typu (*float*),
- określenie właściwości arytmetyki zmiennopozycyjnej.

W kategorii **Tablice i wskazania**
- określenie skutku konwersji danej całkowitej na daną wskazującą i odwrotnie,
- określenie typu danej stanowiącej rezultat użycia operatora *sizeof* oraz typu danej, której wartość określa różnicę wskazań,

W kategorii **Rejestry**
- określenie zasad posługiwania się zmiennymi klasy *register*; w szczególności określenie maksymalnej liczby zmiennych rejestrowych i typu danych, które mogą być przechowywane w rejestrach procesora,

W kategorii **Struktury, unie i pola bitowe**
- określenie sposobu reprezentowania pól unii, w celu uzyskania dostępu do pewnego pola przez nazwę innego pola,
- określenie sposobu rozmieszczenia pól struktury,
- określenie, czy pole typu (*int*) jest traktowane jak pole typu (*signed int*), czy typu (*unsigned int*),
- określenie sposobu rozmieszczenia pól bitowych.

W kategorii **Deklaratory**
- określenie stopnia złożoności deklaratorów.

W kategorii **Instrukcje**
- określenie maksymalnej liczby przedrostków *case* użytych w instrukcji wyboru,

W kategorii **Dyrektywy preprocesora**
- określenie sposobu lokalizowania plików włączanych,
- określenie sposobu interpretowania dyrektyw #*pragma*
- określenie, czy odstępy zawarte w argumencie poddanym działaniu operatora # są zachowane bez zmian, czy są zastępowane pojedynczymi spacjami.

Dokończenie na s. 29

Wielozadaniowa wersja sita Erastotenesa w Adzie

Sito Erastotenesa jest algorytmem znajdowania liczb pierwszych podanym ponad dwa tysiące lat temu przez Erastotenesa z Cyreny. W algorytmie zakłada się początkowo, że wszystkie liczby są pierwsze, a następnie eliminuje się wszystkie wielokrotności kolejnych liczb pierwszych, zaznaczając je jako nie pierwsze. Współcześnie stosuje się ten algorytm jako program wzorcowy do oceny mocy obliczeniowej procesorów w zagadnieniach numerycznych. W latach osiemdziesiątych zmodyfikowano sito Erastotenesa, podając algorytm współbieżny, umożliwiający efektywne wykorzystanie komputerów wieloprocesorowych. Poniżej przedstawiono współbieżną wersję algorytmu zakodowaną w Adzie przy użyciu zadań.

```
------------------------------------------------------------------------
function CALCULATE_PRIMES(MAX_LIMIT : in POSITIVE) return NATURAL is
        type NUMBER_TYPE is (PRIME, NOT_PRIME);
        type NUMBER_ARRAY is array (POSITIVE range <>) of NUMBER_TYPE;
        subtype PRIME_RANGE is NATURAL range 0..MAX_LIMIT;

        POSITIVE_NUMBER : NUMBER_ARRAY(2..MAX_LIMIT+1) := (others=>PRIME);
        NUMBER_PRIMES : PRIME_RANGE := 0;

        task type SLAVE_TASK is
                entry START(PRIME_NUMBER : in POSITIVE);
        end SLAVE_TASK;

        task body SLAVE_TASK is
                INDEX, INCREMENT : POSITIVE;
        begin
                accept START(PRIME_NUMBER : in POSITIVE) do
                        INCREMENT := PRIME_NUMBER;
                        INDEX := 2*PRIME_NUMBER;
                        POSITIVE_NUMBER(INDEX) := NOT_PRIME;
                end START;
                while INDEX <= MAX_LIMIT loop
                        INDEX := INDEX + INCREMENT;
                        POSITIVE_NUMBER(INDEX) := NOT_PRIME;
                end loop;
        end SLAVE_TASK;

        task ERASTOTHENES is
                entry GET_PRIMES(NUMBER_FOUND : out PRIME_RANGE);
        end ERASTOTHENES;

        task body ERASTOTHENES is
                type SLAVE_POINTER is access SLAVE_TASK;
                SLAVE : SLAVE_POINTER;
                NEW_MAX_LIMIT : POSITIVE range 1..MAX_LIMIT;
                PRIME_COUNTER : PRIME_RANGE := 0;
        begin
                NEW_MAX_LIMIT := POSITIVE(FLOAT(MAX_LIMIT)/2.0);
                for LOOP_INDEX in 2..MAX_LIMIT loop
                        if POSITIVE_NUMBER(LOOP_INDEX) = PRIME then
                                PRIME_COUNTER := PRIME_COUNTER + 1;
                                PUT(LOOP_INDEX);        PUT(" ");
                                if LOOP_INDEX <= NEW_MAX_LIMIT then
                                        SLAVE := new SLAVE_TASK;
                                        SLAVE.START(LOOP_INDEX);
                                end if;
                        end if;
                end loop;
                accept GET_PRIMES(NUMBER_FOUND : out PRIME_RANGE) do
                        NUMBER_FOUND := PRIME_COUNTER;
                end GET_PRIMES;
                NEW_LINE;
        end ERASTOTHENES;

begin
        ERASTOTHENES.GET_PRIMES(NUMBER_PRIMES);
        return NUMBER_PRIMES;
end CALCULATE_PRIMES;
------------------------------------------------------------------------
```

Funkcja *CALCULATE_PRIMES*, zrealizowana jako moduł biblioteczny, pobiera jako parametr górną granicę obliczeń, znajduje wszystkie liczby pierwsze w tym zakresie i udostępnia wartość oznaczającą, ile jest tych liczb. Ponadto, funkcja wyświetla wszystkie znalezione liczby.

```
with TEXT_IO, IIO, CALCULATE_PRIMES;
use TEXT_IO, IIO;
------------------------------------------------------------------------
procedure A_PRIMES is
        MAX_NUMBER : POSITIVE range 2..INTEGER'LAST;
begin
        NEW_LINE;
        PUT_LINE("Counting prime numbers up to the limit entered.");
        NEW_LINE;
GET_INPUT:
        loop
        begin
                PUT("Enter limit for finding primes: ");
                GET(MAX_NUMBER);
                exit;
        exception
                when CONSTRAINT_ERROR | DATA_ERROR =>
                        PUT_LINE("Primes are greater than 1;");
                        NEW_LINE;
        end;
        end loop GET_INPUT;
        NEW_LINE;
        PUT_LINE("Wait patiently, I am calculating.");
        PUT_LINE("Number of primes found : "
                & NATURAL'IMAGE(CALCULATE_PRIMES(MAX_NUMBER)));
end A_PRIMES;
------------------------------------------------------------------------
```

Wywołanie funkcji *CALCULATE_PRIMES* powoduje uaktywnienie zadania *ERASTOTHENES*, które powołuje zadanie potomne *SLAVE* i wzywa jego wejście *START*, aby wyeliminować wszystkie wielokrotności liczby 2. Następnie powoływane jest kolejne zadanie potomne *SLAVE*, eliminujące wielokrotności liczby 3. Liczba 4 zostaje ominięta, gdyż została już zaznaczona jako nie pierwsza, a dla liczby 5 powoływane jest nowe zadanie potomne itd. W oryginalnym algorytmie, kolejne zadania *SLAVE* są powoływane aż do przekroczenia liczby będącej pierwiastkiem kwadratowym górnej granicy obliczeń, ponieważ według dowodu przeprowadzonego ponad 600 lat temu przez Fibonacciego osiągnięcie tej liczby oznacza wyeliminowanie przedstawioną metodą wszystkich liczb nie będących pierwszymi. W poniższej implementacji użyto innego ograniczenia.

Errata. Na wydruku funkcji CALCULATE_PRIMES należy zamienić miejscami wiersze pętli **while**, jak poniżej: ımi
POSITIVE_NUMBER(INDEX) := NOT_PRIME; ΛE;
INDEX := INDEX + INCREMENT;

Warunki prenumeraty na lata 1988–1989

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 prenumeratom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu.

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: NBP III Oddział Warszawa 1036-7490--139-11.

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty 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. 27-43-65), 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.

miesięczna		kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
300 zł	60 zł	900 zł	180 zł	1800 zł	360 zł	3600 zł	720 zł

W czterdziestą rocznicę powstania WNT

W bieżącym roku mija 40 rocznica powstania Wydawnictw Naukowo-Technicznych, jednej z czołowych oficyn tego rodzaju w Polsce. Od 40 lat goszczą na półkach księgarskich książki z różnych dziedzin nauki i techniki, podręczniki dla studentów, poradniki dla specjalistów, encyklopedie branżowe, słowniki i leksykony wydane przez WNT. Towarzyszą kolejnym rocznikom studentów różnych specjalności, inżynierom i technikom, kryją w sobie cząstkę historii rozwoju polskiej myśli naukowo-technicznej, w tym – oczywiście również informatyki.

Związek WNT z tą dziedziną sięga początków lat sześćdziesiątych, kiedy jeszcze informatyka nie nazywała się informatyką i kiedy wszelkie działania zmierzające do zainicjowania serii książek z tej dziedziny spotykały się z niechęcią i niedowierzaniem, że ktokolwiek zechce takie książki czytać. Pokonanie tej niechęci wymagało olbrzymiego uporu i zaangażowania tych, którym zależało na polskiej informatyce.

W końcu jednak udało się. W 1965 roku powstała działająca do dziś redakcja informatyki i podstaw techniki. Niedługo potem ukazały się pierwsze książki znanej serii „Informatyka", w ramach której wydano ponad 60 tomów. Zawartość tej serii obejmowała szeroką tematykę, m. in. architekturę komputerów i systemów komputerowych, języki programowania, urządzenia zewnętrzne, zastosowania komputerów w różnych dziedzinach, jak np. w automatyce przemysłowej czy zarządzaniu, modelowanie i symulację komputerową, wyszukiwanie i przetwarzanie informacji. Przeglądając tytuły książek można dostrzec, jak rozwijała się polska informatyka, od czasów gdy komputery nazywano jeszcze maszynami matematycznymi, aż do ery mikroprocesorów. Wśród wydawnictw tej serii jest dużo tłumaczeń, nie brak jednak i pozycji krajowych autorów.

Mniej więcej 11 lat temu pojawiły się pierwsze tomy nowej serii „Biblioteka Inżynierii Oprogramowania", która wyłoniła się z serii „Informatyka". Książki serii BIO są przeznaczone przede wszystkim dla zawodowych programistów z wykształceniem wyższym lub specjalistycznym półwyższym oraz dla studentów kierunków informatycznych. Jej celem jest dostarczenie czytelnikom rzetelnej wiedzy o językach programowania i systemach operacyjnych, podstaw metodycznych i metodologicznych dobrego programowania oraz potrzebnych programistom wiadomości pomocniczych z różnych dziedzin.

Przez jakiś czas książki obu serii ukazywały się równolegle, częściowo pokrywały się też składy komitetów redakcyjnych. Seria „Informatyka" spełniła swoje zadania i wkrótce zaniechano jej kontynuowania. W serii BIO natomiast w ciągu tych 11 lat ukazało się już ok. 50 pozycji, a kilkanaście dalszych jest w przygotowaniu. Największą popularnością cieszyła się książka M. Iglewskiego, J. Madeya i S. Matwina „Pascal. Język wzorcowy". Łączny nakład wszystkich jej wydań wyniósł ok. 30 tys. egzemplarzy.

Warto też wspomnieć o innej serii, od niedawna goszczącej na półkach księgarskich. Seria „Mikrokomputery" w założeniu ma służyć szerokiemu kręgowi odbiorców i jest poświęcona – zgodnie z nazwą – wyłącznie tematyce mikrokomputerowej. Wzrastająca stale liczba mikrokomputerów i ich użytkowników często nie będących informatykami spowodowała ogromne zapotrzebowanie na książki popularne, trafiające w aktualne potrzeby odbiorców szybko rozwijającego się rynku mikrokomputerowego. Książki tej serii są poświęcone nie tyle ogólnym podstawom teoretycznym czy architekturom komputerów, ile konkretnym rozwiązaniom sprzętowym i programowym. W ciągu dwóch lat w tej serii ukazało się kilkanaście książek, dotyczących przeważnie języków programowania i systemów operacyjnych dla popularnych mikrokomputerów.

Oczywiście są również książki wydawane poza seriami, na przykład słowniki i leksykony, będące próbą zebrania i uporządkowania terminologii informatycznej. Wiadomo, że wypowiedzi z tej dziedziny są ciągle jeszcze w dużym stopniu traktowane dość dowolnie, brakuje bowiem dobrych norm ustalających właściwe słownictwo. Widać to szczególnie wyraźnie w różnego rodzaju czasopismach i publikacjach wydawanych przez dość przypadkowe oficyny; nie tylko są ogromne różnice między poszczególnymi wydawnictwami, ale często w ramach tej samej pozycji wydawniczej spotyka się niejednolitą terminologię. WNT stara się lansować w swoich książkach rozsądne ustalenia dotyczące słownictwa informatycznego, chroniące przed „zaśmiecaniem" naszego języka terminami angielskimi i przedziwnymi neologizmami.

Dla środowiska zawodowego łatwy dostęp do dobrej literatury fachowej jest rzeczą niezwykle istotną. Działalność Wydawnictwa i współpracujących z nim komitetów redakcyjnych poszczególnych serii zmierza do tego, by dostarczyć czytelnikom jak najwięcej książek na jak najwyższym poziomie, inspirować do pisania krajowych autorów i tłumaczyć książki wydane w innych krajach. Oczywiście, jak wszędzie, jest wiele ograniczeń i przeszkód: nie zawsze można wydać książkę tak szybko, by znalazła się na rynku w chwili największego na nią zapotrzebowania, nie zawsze można uzyskać zgodę na wydanie tłumaczenia wybranej pozycji zagranicznej, nie zawsze nakład może być wystarczający. W tej sytuacji cenna jest każda inicjatywa zmierzająca do usprawnienia działalności.

Jedną z takich inicjatyw była decyzja o wprowadzeniu w Wydawnictwie techniki mikrokomputerowej, która umożliwiła przyspieszenie cyklu produkcji książki. Zaczęło się skromnie, od jednego komputera ze zwykłą drukarką 9-igłową. Przymknięto oczy na nie najlepszą jakość techniczną, wiedząc że jest to cena za krótki cykl produkcji – i w ten sposób powstały pierwsze książki (przypominające raczej skrypty), realizowane nietradycyjną metodą. Dziś w WNT jest oddzielny dział, w którym przygotowuje się publikacje metodą komputerową. Do dyspozycji jest oczywiście znacznie lepszy sprzęt i oprogramowanie, dzięki czemu efekt końcowy jest już porównywalny z profesjonalnym składem.

Autorzy przynoszący swoje teksty na dyskietkach nie są już traktowani jak natręci czy dziwacy. Znikła konieczność kilkukrotnego przepisywania tekstu i związana z tym nieuchronna możliwość wprowadzania coraz to nowych błędów. Oszczędność czasu jest ewidentna, ale jeszcze istotniejsze jest częściowe uniezależnienie się od drukarni. Skład bowiem jest tym wąskim gardłem, które sprawia, że książka leży w drukarni czasem rok, czasem dwa lata, a zdarza się, że nawet dłużej. Wykonanie wydruków na drukarce laserowej i przesłanie ich do drukarni do bezpośredniej reprodukcji pozwala wyeliminować skład i przez to przyspieszyć wydanie książki.

Nie jest to oczywiście jedyne rozwiązanie ani wcale nie najlepsze. Uzyskana jakość techniczna tak przygotowanej książki, aczkolwiek w zupełności wystarczająca dla przeciętnego czytelnika, może wydać się niedostateczna dla bardziej wytrawnych znawców. Warto więc podjąć także próby innego rodzaju, np. przygotowanie dyskietek z tekstem i przesłanie ich do drukarni, gdzie będą następnie odczytane w urządzeniach do fotoskładu. Z różnych względów trwałoby to nieco dłużej, ale nagrodą byłaby jakość – najlepsza jaką można uzyskać, gdyż określona przez użycie profesjonalnego fotoskładu.

Dokąd prowadzi ta droga, zobaczymy w najnowszych publikacjach. Przy okazji takiego jubileuszu należałoby natomiast życzyć Wydawnictwu i jego czytelnikom, aby cykl produkcyjny książki – od dostarczenia przez autora do jej ukazania się na półkach księgarskich – trwał np. pół roku i aby było to nie tylko możliwe, ale i normalne; przynajmniej, jeśli chodzi o książki z tak dynamicznie rozwijającej się dziedziny, jaką jest informatyka.

Redakcja

Biblioteki procedur Turbo Pascala

Power Tools Plus, wersja 2.00, firmy Blaise Computing i Turbo Extender, wersja 1.04, firmy Turbopower, Software to biblioteki procedur Turbo Pascala wydatnie zwiększające wydajność komputerów IBM PC/XT, PC/AT i zgodnych, pracujących pod kontrolą systemu operacyjnego PC-DOS 2.0 lub o numerze większym. Pakiety testowano na komputerze IBM PC/XT z pamięcią 512 KB, z koprocesorem 8087, dyskiem stałym 20 MB i napędami dysków elastycznych 360 KB, pod nadzorem systemu PC-DOS 3.1. Liczba buforów systemowych określona w pliku konfiguracyjnym wpływa na wykonanie programów i operacje we-wy: ustalono liczbę buforów na 16.

POWER TOOLS PLUS

Mimo że pakiet Power Tools Plus 2.00 może współpracować z Turbo Pascalem 2.0, to zaleca się wersję 3.0. Najmniejsza pamięć wymagana do współpracy jest taka sama jak dla Turbo Pascala – 64 KB. Zaleca się użycie stałego dysku, ale nie jest on wymagany. W bibliotece zawarto funkcje: manipulowania napisami, procedury obsługi ekranu, zarządzania oknami, obsługi menu, sterowania klawiaturą, korzystania z funkcji DOS, obsługi plików, zarządzania skorowidzami, zarządzania pamięcią, sterujące programem i obsługujące przerwania.

Wśród funkcji napisowych zwracają uwagę **FillStr, LeftStr, RightStr, MidStr** i **SubStr** ze względu na szybkość wykonania i stworzenie w Pascalu odpowiedników funkcji Basica: **STRING$, TIME$, DATE$, LEFT$, RIGHT$** i **MID$**, jak również wyrównywanie napisów, konwersje numeryczne i konwersje między małymi i dużymi literami. Zmierzone czasy dla części składowych napisów o zmiennej długości obrazują efekt manipulacji różną liczbą znaków. Funkcja **RightStr** wykonuje się wolniej wraz ze zmniejszeniem liczby znaków, natomiast wszystkie inne funkcje zgodnie z oczekiwaniem – wykazują tendencje przeciwne.

Procedury pakietu Power Tools Plus obsługujące ekran, sterujące kursorem i grafiką pozwalają na pracę w trybie monochromatycznym, CGA, PCjr i EGA. Niektóre z tych procedur dublują wewnętrzne mechanizmy Turbo Pascala w celu przystosowania się do różnych kart graficznych. W zależności od typu wykorzystywanego monitora można otrzymać kilka ekranów tekstowych. Procedura biblioteczna rozpoznaje typ monitora i dokładną liczbę wyświetlanych stron tekstowych z pamięci. Można podczas oglądania jednej strony zapisywać drugą, ponieważ procedura obsługuje jednocześnie bieżącą stronę i stronę wyświetlaną (aktywną). Szybkość pobierania i przesyłania danych na ekran można zwiększyć wykorzystując bezpośredni dostęp do pamięci ekranu.

Procedury obsługi okien rozszerzają możliwości tworzenia okien tekstowych w Turbo Pascalu o opcje pracy z kartą graficzną EGA, tworzenie, usuwanie i manipulowanie wieloma oknami (z zaznaczonymi granicami). Uzyskuje się dodatkową możliwość sterowania kursorem, atrybutami obrazu i wykonywania operacji we-wy. Procedury obsługi menu zawierają specjalny rodzaj okien. Tworzenie menu odbywa się w trzech etapach: tworzenie deskryptora menu, specyfikacja wyglądu menu i wywołanie procedury pokazującej menu i udostępniającej dokonany wybór.

Procedury sterujące klawiaturą wykonują przeszukiwanie klawiatury i specjalne działania wejściowe odporne na błędy. Przy przeszukiwaniu klawiatury jest podawany kod rozszerzony (generowany naciśnięciem klawisza funkcyjnego lub strzałkowego), a następnie sprawdzony i zapisany stan takich klawiszy jak CAPS-LOCK, NUM-LOCK, SCROLL-LOCK lub kombinacji klawiszy ALT-SHIFT. Procedury wykonują również czyszczenie bufora klawiatury i dwustronne przesyłanie znaków między buforem i programem aplikacyjnym. Funkcje zabezpieczające przed błędami umożliwiają podanie w programie prośby o wpisanie liczb całkowitych lub rzeczywistych; jeśli podane dane nie spełniają definicji liczb całkowitych lub rzeczywistych, to wyświetla się komunikat. Te funkcje zabezpieczają przed zatrzymaniem programu w wypadku podania błędnych danych, co może być denerwujące przy wprowadzaniu dużej tablicy.

Procedury korzystania z funkcji DOS pozwalają programom użytkowym Turbo Pascala sprawdzić stan DOS-u i sterować jego środowiskiem. Można ustawiać czas i datę w systemie, sprawdzić numer wersji DOS, sprawdzić czy system operacyjny nie jest w krytycznym stanie nieprzerwaniowym, sprawdzać i ustawiać parametry środowiska DOS i ich bieżący stan. Wykonując te procedury wpływa się tylko na kopię środowiska DOS i dowolne programy niższego rzędu (wykonywane z wnętrza innych) inicjalizowane przez aplikacje – nie wpływa się natomiast na oryginalne środowisko DOS związane z plikiem COMMAND. COM. Można zainstalować pracujący pod kontrolą DOS 3.0 spooler PRINT, po czym wstawić i usuwać z kolejki do drukowania pliki tekstowe.

Procedury obsługi plików pakietu Power Tools Plus umożliwiają manipulowanie plikami nietekstowymi: tworzenie pliku, otwieranie, zamykanie i wykonywanie operacji we-wy. Przy dostępie swobodnym do pliku wskaźnik jest przemieszczany względnie lub bezwzględnie. Można ustawiać lub odczytywać datę i czas utworzenia pliku, atrybuty pliku i obszar przesyłania danych DTA. Wykorzystywane procedury różnią się od analogicznych Turbo Pascala tym, że odwołanie do pliku odbywa się za pomocą numeru logicznego pliku. Dodatkowo operacje we-wy są wykonywane z wykorzystaniem wskaźników i liczników rejestrujących źródło i przeznaczenie danych. Procedury obsługi plików udostępniają prawa dostępu do pliku charakterystyczne dla sieci i dla systemu operacyjnego DOS, co oznacza możliwość blokowania i odblokowania części lub całego pliku.

Następna grupa procedur obsługuje skorowidze. Niektóre z nich tworzące, usuwające i zmieniające skorowidze dublują polecenia systemu operacyjnego; inne wykonują bardziej wyrafinowane zadania, takie jak: zmiana nazwy skorowidza, podanie lub ustawienie etykiety nośnika, przeszukiwanie skorowidza. Obydwie grupy procedur udostępniają właściwości bardzo pożyteczne dla zaawansowanych programistów.

Procedury zarządzające pamięcią udostępniają odpowiednie usługi systemu operacyjnego. Procedury pozwalają określić wielkość zainstalowanej pamięci, jak również wielkośc pamięci dostępnej. Można również sprawdzić wielkość pamięci przeznaczonej na środowisko Turbo Pascala obejmujące obszar programu, obszar danych statycznych, najmniejszy obszar stosu i największy obszar stosu i stogu. Za pomocą tych procedur i wskaźników można przydzielać i zmieniać bloki pamięci systemu operacyjnego. Procedury umożliwiają wykrycie rozszerzonej pamięci zainstalowanej w mikrokomputerze IBM PC/AT lub przesłania danych między pamięcią główną i rozszerzoną.

Procedury sterujące programem w celu wywołania procesów pochodnych współpracują z procedurami zarządzania pamięcią. Po zakończeniu wykonania procedury mogą rezydować w pamięci RAM, co przypomina inne rezydujące aplikacje, takie jak SideKick, SuperKey lub ProKey. Z programu aplikacyjnego można wykonać nawet polecenie systemu DOS – procedura biblioteczna ładuje kopię COMMAND.COM jako proces pochodny i wykonuje podane polecenie.

W pakiecie znajduje się zestaw procedur niskiego poziomu do obsługi przerwań. Dzięki nim można wywołać procedury rezydujące w pamięci RAM i tworzyć programy przedefiniowania klawiatury, takie jak SuperKey. Pakiet zawiera **MKEY**, prosty program tego rodzaju, stanowiący przykład wykorzystywania procedur obsługi przerwań wspólnie z procedurami sterowania programem. Ponadto, można włączać i wyłączać przerwania, ustawiać lub sprawdzać wektory przerwań itd.

W pakiecie Power Tools Plus zawarto kilka programów usługowych. Jeden z nich, sterujący drukiem programów źródłowych ma możliwości doboru wielkości strony i marginesu. Następny, o nazwie **INCLUDE**, generuje spis dyrektyw *include* i umieszcza na pliku źródłowym. Dokonuje sprawdzenia prawidłowości wywołań hierarchicznie powiązanych procedur i odwzorowuje te zależności. **PROCPAK** usuwa komentarze, znaki tabulacji
i puste linie, tym samym zmniejszając objętość pliku źródłowego, który zajmie mniej miejsca na dysku. **CLOCK** to rezydujący zegar, który można zainstalować w kolorze, z komunikatem alarmowym, na dowolny przedział.

Podręcznik liczący 310 stron zawiera kilka dodatków poświęconych pobieżnemu omówieniu procedur i powiązań między nimi, ważnych typów danych i zmiennych, a nawet odpowiedzi na szczególnie trudne pytania.

TURBO EXTENDER

Turbo Extender usuwa ograniczenie Turbo Pascala 3.00 co do wielkości programu – 64 KB. Ponieważ pakiet jest ukierunkowany na duże systemy, to nie ma żadnej wzmianki o pamięci minimalnej lub zalecanej. W Turbo Extender zamiast łańcuchowania lub nakładkowania programów wykorzystuje się szybszy sposób – tworzenie oddzielnie kompilowanych programów modułowych. Tym samym, przy nanoszeniu zmian należy skompilować tylko tę część, w której nastąpiła zmiana. Pakiet zawiera bibliotekę dołączanych plików o nazwie **BIGTURBO**, dzięki której można tworzyć moduły Pascala obsługujące procedury wywoływane przez granice modułów.

Biblioteka **BIGTURBO** zawiera następujące programy usługowe:

SHELLGEN tworzący moduły źródłowe Pascala przez dodanie do programu źródłowego dyrektyw definiujących granice modułu,
BIGMAKE znajdujący zmienione moduły a następnie ponownie je kompilujący (podobny do **make** w Unixie),
EXPORTER sprawdzający odwołania międzymodułowe i nadzorujący jednolitość struktur BIGTURBO,
BUILDEXE dający możliwość utworzenia pojedyńczego pliku wykonawczego EXE.

Turbo Extender obsługuje również tablice o objętości przekraczającej 64 KB. Biblioteka **BIGARRAY** za pomocą wirtualnego systemu ze stronicowaniem obsługuje pięć modeli pamięci. Te modele pamięci obejmują:
- tablice wykorzystujące pamięć RAM poza przestrzenią roboczą Turbo Pascala do przechowywania dużych tablic,
- duże tablice do przechowywania macierzy rzadkich i wskaźników pozwalających pominąć puste elementy macierzy,
- tablice dyskowe (tj. duże tablice są przechowywane na dysku),
- wirtualne tablice dyskowe, które są podobne do tablic dyskowych, ale pozwalają w trakcie kompilacji lub wykonania określić rozmiar tablicy,
- tablice rezydujące w pamięci rozszerzonej, do przechowywania dużych tablic według specyfikacji pamięci rozszerzonej Lotus/Intel/Microsoft.

BIGARRAY zawiera procedury tworzenia, inicjalizacji, usuwania i manipulacji tablicami. Procedury dla każdego typu pamięci są podobne i pozwalają na przejście między modelami pamięci po niewielkich zabiegach redakcyjnych. Korzystając z biblioteki BIGARRAY należy oczekiwać spadku szybkości przetwarzania tablic i macierzy. Oczywiście sama możliwość działania na dużych tablicach jest niepodważalną zaletą tego pakietu.

Przeprowadzono badanie szybkości przetwarzania dużych tablic w pamięci, tj. poza segmentem danych Turbo Pascala 3.0. W wypadku rzeczywistych macierzy o wymiarach 50×50, ich mnożenie w segmencie danych było 2,25 raza szybsze niż za pomocą procedur Turbo Extendera. Mnożenie tablic o 75 elementach było szybsze 4,61 raza. Jednakże Turbo Pascal nie może wykonywać mnożenia większych tablic w segmencie danych. Przetestowano pakiet na mnożenie w pamięci RAM tablic o wymiarach 120×120 (całkowitych i rzeczywistych). Mnożenie tablicy rzeczywistej zajęło więcej niż 36 minut, a całkowitej około 22 minut. Oczywiście w Turbo Pascalu nie ma co próbować przetwarzania takich tablic.

Turbo Extender zawiera również inne programy usługowe do analizy programów nakładkowych, wykorzystania pamięci podręcznej dla dysku i szyfrowania programu źródłowego. Program szyfrujący PCRYPT.COM ma interesującą właściwość – przekształca pierwotną postać pliku źródłowego na postać bardzo nieczytelną. Pliki w postaci zaszyfrowanej są kompilowane szybciej, ponieważ odpowiadają sposobowi przechowywania tablic danych w Turbo Pascalu. Testy przeprowadzone na programach o wielkości ok. 30 KB potwierdzają znaczne zredukowanie wielkości pliku (w granicach od 25% do 40%). Jednakże oszczędności na czasie kompilacji były niewielkie (około 1 s. na program). Program źródłowy zapisywany w kodzie ASCII kompiluje się szybciej, ale program szyfrowany jest nieco krótszy i stąd ta niewielka oszczędność. Oczywiście, zmniejszenie objętości pliku jest samo w sobie bardzo pożyteczne; nawet gdy zysk na czasie kompilacji jest niewielki, to szyfrowanie plików źródłowych można uznać za pożyteczną właściwość PCRYPT.COM.

. . .

Power Tools Plus jest produktem programistycznym zdradzającym dobre opanowanie rzemiosła przez twórców, wspartym bardzo dobrą dokumentacją. Nowa wersja jest rozszerzeniem oryginalnego pakietu Power Tools i ma dużo lepszą bibliotekę. Omawiany pakiet można śmiało polecić programistom poszukującym solidnej, dobrze zaprojektowanej biblioteki procedur.

Turbo Extender powinien przydać się tym programistom, którym przeszkadza ograniczenie Turbo Pascala na wielkość programu lub danych (64 KB). Dzięki niemu można pisać dużo większe programy lub operować większymi tablicami.

Obydwa pakiety są komplementarne w stosunku do siebie.

Opracował
M. KUC
na podstawie „Byte"

Turbo C

Programowanie przenośne

dokończenie ze s. 25

W kategorii **Funkcje**
- określenie rozwinięcia makrodefinicji *NULL*,
- określenie dziedzin funkcji,
- określenie domniemań w obsłudze sygnałów,
- określenie zasad buforowania w operacjach wejścia-wyjścia,
- określenie, czy możliwe jest utworzenie pliku nie zawierającego danych,
- określenie zasad tworzenia poprawnych nazw plików,
- określenie, czy więcej niż jeden plik może odwoływać się do tego samego pliku zewnętrznego,
- określenie skutków użycia konwersji *%p*,

W kategorii **Rozszerzenia**
- określenie, czy funkcja *main* może być wywoływana z argumentem odpowiadającym parametrowi *char *envp []*;
- określenie, czy identyfikatory mogą zawierać dodatkowe znaki, takie jak np. $,
- określenie, czy poprawne jest dokonywanie zmian ciągu znaków reprezentowanego przez literał napisowy,
- określenie, czy dopuszczalne jest posługiwanie się dodatkowymi typami, takimi jak np. *long long int*,
- określenie, czy w celu potraktowania danych jak kodu, jest poprawne przekształcenie wskazania danej we wskazania funkcji,
- określenie, czy jest poprawne posługiwanie się wstawkami asemblerowymi, np. rozpoczynającymi się od słowa kluczowego *asm*.

Sztuczna inteligencja

A. Kobsa, W. Wahlster (Eds.): User Models in Dialog Systems. Springer-Verlag. Berlin, 1988

Modele użytkownika stały się ostatnio obiektem dużego zainteresowania badaczy sztucznej inteligencji, zajmujących się tworzeniem systemów konwersacyjnych. Okazało się, że systemy konwersacyjne mogą zapewnić elastyczną komunikację z użytkownikiem tylko wówczas, gdy posługują się modelem zawierającym założenia co do umiejętności użytkownika, jak również celów i planów jego działania w trakcie wykorzystywania danego systemu. Badania w dziedzinie modeli użytkownika koncentrują się na sposobach automatycznego generowania, reprezentowania i wykorzystywania założeń tego rodzaju w czasie interakcji z użytkownikiem. Książka jest pierwszą publikowaną pracą z zakresu budowy modeli użytkownika i składa się z artykułów przygotowanych przez znanych badaczy pracujących w tej dziedzinie. Artykuły zostały podzielone na cztery grupy. Pierwsza część stanowi ogólne wprowadzenie do dziedziny; omówiono w niej problemy oraz techniki konstruowania modeli. Części druga i trzecia zawierają opisy ośmiu systemów opartych na modelach użytkownika. W czwartej części przedstawiono ograniczenia obecnych systemów i zaproponowano sposoby ich przezwyciężenia. Książka zawiera bogatą bibliografię.

W. Bibel, P. Jorrand (Eds): Fundamentals of Artificial Intelligence (Second Edition). Springer-Verlag. Berlin, 1987

Książka składa się z siedmiu specjalnie opracowanych wykładów, wygłoszonych na Zaawansowanym Kursie Sztucznej Inteligencji, który przeprowadzono w Vignieu, we Francji, w lipcu 1985 r. Artykuły zostały pomyślane jako materiał dydaktyczny, tak więc książka stanowi cenne wprowadzenie do podstawowych zagadnień sztucznej inteligencji. W pierwszej części Delgrande i Mylopoulos analizują pojęcie wiedzy i jej różne reprezentacje. Druga część książki poświęcona jest przetwarzaniu wiedzy. Artykuł Hueta udowadnia, że zarówno obliczenia, jak i wnioskowanie czy dedukcja są w istocie różnymi aspektami tego samego zjawiska. Stickel wprowadza czytelnika w najważniejsze zagadnienia dedukcji z podkreśleniem roli mechanizmów rezolucji. Artykuł Biermanna dotyczy technik wnioskowania stosowanych we wnioskowaniu indukcyjnym, rozwiązywaniu problemów na podstawie przykładów i – maszynowym uczeniu się. Bibel omawia pewne aspekty przetwarzania wiedzy, które mogą znaleźć zastosowanie we wnioskowaniu zdroworozsądkowym. Trzecia część książki dotyczy programowania logicznego i funkcjonalnego. Jorrand opisuje język FP2, umożliwiający zarówno programowanie funkcjonalne, jak i równoległe. W ostatnim artykule Shapiro przedstawia aktualny stan prac nad współbieżnym Prologiem.

L. Pun: Industrial Artificial Intelligence Systems. Plenum Press, New York, 1988

Zastosowanie technik sztucznej inteligencji w przemyśle ma na celu usprawnienie procesów produkcyjnych. Niezbędna jest do tego nie tylko dobra znajomość określonej gałęzi przemysłu, technik automatyzacji i teorii sterowania, lecz także umiejętność budowy modeli obliczeniowych procesów sterowania i procesów podejmowania decyzji. Książka stanowi syntezę wiedzy na temat tzw. przemysłowych systemów sztucznej inteligencji. Podaje definicje podstawowych pojęć, opisy różnych konfiguracji, zasady projektowania, teoretyczne podstawy rozwiązywania problemów oraz sposoby implementowania modeli obliczeniowych. W książce omówiono narzędzia analizy i budowy takich systemów, a w szczególności grafy określające zależności między czynnościami i rezultatami tych czynności, logikę predykatów pierwszego rzędu oraz struktury danych oparte na hipergrafach. Przedstawione pojęcia i metody zilustrowano praktycznymi przykładami.

D.T. Pham (Ed.): Expert Systems in Enginerring. Springer-Verlag, Berlin, 1988

Książka zawiera zbiór artykułów opisujących możliwości wykorzystania technik systemów ekspertowych w przemyśle. Przykładowe zastosowania dotyczą kontroli procesów technologicznych, budownictwa, produkcji urządzeń elektrycznych i elektronicznych, budowy maszyn oraz automatyzacji produkcji. Omówione systemy ekspertowe rozwiązują następujące rodzaje zadań: nadzór procesu, diagnozowanie uszkodzeń, przewidywanie, planowanie akcji i projektowanie.

(MM)

Kto jest kim w IFIP

James Finch

Członek zarządu IFIP, James Finch, jest konsultantem specjalizującym się w zagadnieniach poufności systemów przetwarzania danych. Obecnie jest dyrektorem konsultingowej firmy Cerebrus Computer Security, działającej na terenie Ameryki Północnej i Europy.

James Finch urodził się w Toronto, gdzie w 1955 r. ukończył studia uniwersyteckie i rozpoczął karierę zawodową podejmując pracę w dużych przedsiębiorstwach. W 1967 r. odszedł z pracy w przemyśle i założył usługowe przedsiębiorstwo komputerowe. Od 1974 r., kiedy zaangażował się w badania nad oszustwami i nadużyciami komputerowymi, specjalizuje się w konsultowaniu tych zagadnień. Firma J. Fincha prowadzi także konsultacje nt. planowania strategicznego i polityki dotyczącej zarządzania systemami informacyjnymi – głównie dla instytucji przemysłowych i rządowych.

James Finch jest członkiem wielu stowarzyszeń zawodowych, m. in. Kanadyjskiego Towarzystwa Przetwarzania Informacji (*Canadian Information Processing Society, CIPS*), gdzie pełni wiele funkcji, wśród nich – przewodniczącego. W latach 1978–1979 organizował Krajowe Seminarium CIPS nt. bezpieczeństwa i kontroli przetwarzania informacji. W 1980 r. został prezesem-założycielem specjalnej grupy CIPS ds. poufności. W maju ub.r. stowarzyszenie to uhonorowało go pierwszym odznaczeniem CIPS Contribution Award.

J. Finch zapoczątkował udział w pracach IFIP jako przewodniczący Komitetu Organizacyjnego Kongresu IFIP w 1977 r., w Toronto. Od 1980 r. jest przedstawicielem CIPS w IFIP. Pełnił w tej organizacji wiele funkcji. Od 1980 do 1986 r. był członkiem zarządu, do którego został ponownie wybrany w 1987 r. Jest przewodniczącym Komitetu ds. Wyboru Miejsca Kongresów i Komitetu ds. Zaleceń Organizacyjnych Kongresów. Jest również mężem zaufania (ang. cognizant officer) Komitetów Technicznych TC6 i TC10. J. Finch był współzałożycielem Komitetu Technicznego ds. Poufności i Ochrony Systemów Przetwarzania Informacji (TC11), jak też przewodniczącym międzynarodowego komitetu programowego pierwszej konferencji zorganizowanej przez ten Komitet – IFIP/Sec'83. Obecnie jest przewodniczącym grupy roboczej ds. zabezpieczeń (WG 11.1).

James Finch i jego żona Pat mają pięcioro dzieci, w wieku od 20 do 32 lat, jak również trójkę wnucząt. W czasie wolnym żeglują, lubią również narty i podróże.

Oprac. MK
na podst. IFIP Newsletter

Pawłowski M.: Przegląd układów szybkiego mnożenia

INFORMATYKA 1989, nr 4, s. 1

Charakterystyka rozwiązań konstrukcyjnych oraz sposobu działania obecnie stosowanych w komputerach układów szybkiego mnożenia.

Pawłowski M.: Survey of high speed multiplication circuits

INFORMATYKA 1989, No. 4, p. 1

Characteristics of constructional solutions and operating methods in different high speed multiplication circuits, which are applied in contemporary computers.

Pawłowski M.: Ein Übersicht von Schaltungen für schnelle Multiplizierung

INFORMATYKA 1989, Nr. 4, S. 1

Eine Charakteristik von Konstruktionslösungen und Wirkungsweise der verschiedenen, in heutigen Computern angewendeten Schaltungen für schnelle Multiplizierung.

Koziarski K.: Technologia konwersji oprogramowania cobolowego na IBM PC

INFORMATYKA 1989, nr 4, s. 6

Charakterystyka wersji języka Cobol stosowanej na komputerach klasy IBM PC oraz omówienie metody przenoszenia istniejącego w tym języku oprogramowania komputerów klasy Odra i Riad, a także eksploatacji systemów przetwarzania danych w zestawie Odra–PC.

Koziarski K.: A technology of Cobol software conversion for IBM PC

INFORMATYKA 1989, No. 4, p. 6

Characteristics of the Cobol version for IBM PC and discussion of transportation methods for existing Cobol software from Odra and Riad computers, as well as of data processing system's operation on Odra with PC configuration.

Koziarski K.: Eine Technologie der Konversion von Cobol-Software auf IBM PC

INFORMATYKA 1989, Nr. 4, S. 6

Eine Charakteristik der auf IBM PC angewendeten Cobol-Version und eine Besprechung der Methode zur Übertragung bestehender und auf Odra und ESER-Computern ausgenützten Cobol-Software, sowie von Betrieb der EDV-Systeme auf gemeinsamen Odra und PC-Konfiguration.

Nowiński W.: Algorytmy rekonstruowania obrazów za pomocą metod transformacyjnych

INFORMATYKA 1989, nr 4, s. 10

Charakterystyka algorytmów stosowanych do komputerowego rekonstruowania obrazów oraz sposób ich implementacji.

Nowiński W.: Algorithms for image reconstruction using transformation methods

INFORMATYKA 1989, No. 4, p. 10

Characteristics of algorithms and their implementation methods which are applied for computerized image reconstruction.

Nowiński W.: Die Algorithmen für Bildrekonstruierung mit Hilfe von Transformationsmethoden

INFORMATYKA 1989, Nr. 4, S. 10

Eine Charakteristik von Algorithmen und Methode ihrer Implementierung bei rechnergestützten Bildrekonstruierung.

Simon H.A.: Procesy przeszukiwania i wnioskowania w rozwiązywaniu problemów (2)

INFORMATYKA 1989, nr 4, s. 12

Druga część artykułu z dziedziny badań nad sztuczną inteligencją, zawierająca omówienie oraz porównanie różnych systemów przeszukiwania przestrzeni rozwiązań.

Simon H.A.: Search and reasoning in problem solving (2)

INFORMATYKA 1989, No. 4, p. 12

Second part of the paper from the field of AI research, which includes discussion and comparison of different systems for solution's space search.

Simon H.A.: Suchungs- und Schlussfolgerngsprozesse bei Lösung von Problemen (2)

INFORMATYKA 1989, Nr. 4, S. 12

Zweiter Teil eines Artikels aus dem Bereich der künstlichen Intelligenz, der eine Besprechung und Vergleichung von verschiedenen Systemen zur Durchsuchung des Lösungsraumes.

Rybnik J., Solak J.: Stacje robocze Sun-3 (2). Oprogramowanie

INFORMATYKA 1989, nr 4, s. 16

Druga część artykułu na temat charakterystycznych cech stacji roboczych, zawierająca omówienie wszystkich składników oprogramowania stacji roboczych rodziny Sun-3.

Rybnik J., Solak J.: Sun-3 workstations (2). The software

INFORMATYKA 1989, No. 4, p. 16

Second part of the paper on characteristic features of workstations, which includes discussion of all software components for the Sun-3 workstation family.

Rybnik J., Solak J.: Sun-3-Arbeitsplatzcomputer (2). Die Software

INFORMATYKA 1989, Nr. 4, S. 16

Zweiter Teil des Artikels über charakteristische Eigenschaften der Arbeitsplatzcomputer, der eine Besprechung aller Softwarekomponenten von Sun-3-Arbeitsplatzcomputerfamilie umfasst.

Hornowski M.: Metoda instalowania bardzo dużych programów

INFORMATYKA 1989, nr 4, s. 20

Charakterystyka opracowanej w Instytucie Technologii Elektronowej w Warszawie metody efektywnego instalowania bardzo dużych programów napisanych w językach wysokiego poziomu.

Hornowski M.: A method for installation of very large programs

INFORMATYKA 1989, No. 4, p. 20

Characteristics of in the Electron Technology Institute in Warsaw elaborated method for improving of installation of very large and in high level language written programs.

Hornowski M.: Eine Methode zur Installierung der sehr umfangreichen Programme

INFORMATYKA 1989, Nr. 4, S. 20

Eine Charakteristik von im Institut für Elektrontechnologie in Warschau erarbeiteten Methode, die Installierung der sehr umfangreichen und in Hochsprachen geschriebenen Programme rationalisiert.

Ossowska S.: Dobór metody projektowania systemów informatycznych zarządzania

INFORMATYKA 1989, nr 4, s. 22

Charakterystyka porównawcza dwóch podstawowych metod projektowania systemów informatycznych zarządzania oraz omówienie problemu doboru właściwej metody.

Ossowska S.: Choice of method for management information system designing

INFORMATYKA 1989, No. 4, p. 22

Comparative characteristics of the basic methods for management information system designing and discussion of suitable method selecting.

Ossowska S.: Auswahl einer Methode für Projektierung der EDV-Verwaltungssystemen

INFORMATION 1989 Nr. 4, S. 22

Eine Vergleichscharakteristik der zwei grundlegenden Projektierungsmethoden für EDV-Verwaltungssysteme und eine Besprechung von Auswahl der passenden Methode.

Bielecki J.: Turbo C – programowanie przenośne

INFORMATYKA 1989, nr 4, s. 24

Charakterystyka zasad oraz przykłady konstrukcji programowania przenośnego w języku Turbo C.

Bielecki J.: Turbo C – portable programming

INFORMATYKA 1989, No. 4, p. 24

Characteristics of principles and construction example of portable programming in the Turbo C language.

Bielecki J.: Turbo C – übertragbare Programmierung

INFORMATYKA 1989, Nr. 4, S. 24

Eine Charakteristik von Grundsätzen und Beispiele von Konstruktionen der übertragbaren Programmierung in der Turbo C-Sprache.

Propozycja do dyskusji

Informatyka w szkołach okazją do uproszczenia ortografii

Wprowadzenie do szkół informatyki i sprzętu komputerowego oraz języków programowania z napisami (instrukcjami) w języku polskim jest dobrą okazją, ale i ostatnim terminem, do względnie taniego uporządkowania ortografii języka polskiego – odrzucenia zbędnego balastu w postaci podwójnych oznaczeń tych samych dźwięków, wprowadzenia uzupełnień w alfabecie itp. Uprościłoby to naszą pisownię i zbliżyłoby ją do pisowni języków słowiańskich, opartych na alfabecie łacińskim, zwłaszcza do języków: łużyckiego, czeskiego i słowackiego (umożliwiłoby naturalne „schodzenie się" języka polskiego z tymi językami). Stałaby się ona dzięki temu łatwiejsza do opanowania przez dzieci (odciążenie programów szkolnych), nas samych i cudzoziemców.

Proponuję również formalne włączenie do alfabetu polskiego i pełne wykorzystanie wszystkich znaków podstawowego alfabetu łacińskiego, które w istocie są powszechnie używane w szkole (zwłaszcza na lekcjach z przedmiotów ścisłych, poczynając od szkoły podstawowej), literaturze zawodowej, a nawet w prasie. W informatyce używane są one nawet na jej najbardziej elementarnym poziomie. Pismo stanie się przez to bardziej zwięzłe (teksty techniczne o ok. 8% – aspekt ekonomiczny).

A oto szczegółowe propozycje:

1) wprowadzenie dwóch nowych znaków $\dot{c}$ i $\dot{s}$ na oznaczenie dźwięków *cz* i *sz* (zasada oddzielnych znaków dla pochodnych dźwięków – podstawowego, zmiękczonego i utwardzonego):
$c \rightarrow \acute{c} \rightarrow \dot{c}$ (zamiast cz)
$s \rightarrow \acute{s} \rightarrow \dot{s}$ (zamiast sz)
$z \rightarrow \acute{z} \rightarrow \dot{z}$ (te znaki już są);

2) uproszczenie ortografii:
$\dot{z}$, $rz \rightarrow \dot{z}$
h, $ch \rightarrow h$
(obydwa znaki – jedno- i dwuliterowy oznaczają ten sam dźwięk);

3) formalne włączenie do alfabetu polskiego rzeczywiście używanych łacińskich znaków *x*, *v* i *q*;

4) ujednolicenie pisowni z innymi językami:
$w \rightarrow v$

5) pełne wykorzystanie włączonych znaków przez przypisanie im dźwięków:
$ks \rightarrow x$ (przykład pisowni – *Xavery*)
$kw \rightarrow q$ (w proponowanej pisowni *kv*);

6) przypisanie nowego dźwięku znakowi *w*:
$wu \rightarrow w$ (w proponowanej pisowni *vu*).

Liczę na poparcie moich koncepcji przez zawodowych informatyków, zwłaszcza producentów sprzętu komputerowego, ale również wszystkich użytkowników tych systemów, no i, oczywiście, młodzież szkolną i nauczycieli. Sądzę, że wspólnym działaniem udałoby nam się pozyskać językoznawców, księgarzy i dziennikarzy oraz władze oświatowe do wprowadzenia sugerowanych modyfikacji w interesie nas wszystkich.

(WM)

Od Redakcji. Uważni Czytelnicy zapewne zorientowali się, że niniejszą propozycję przedstawiamy w numerze kwietniowym.

Ceny ogłoszeń

Od 1 stycznia 1989 r. obowiązują następujące ceny materiałów reklamowych publikowanych na łamach INFORMATYKI:

Ogłoszenia
– ogłoszenia czarno-białe, artykuły reklamowe i informacje naukowo-techniczne (biuletyny) zależnie od objętości: cała strona – 70 tys., 3/4 s. – 60 tys., 2/3 s. – 55 tys., 1/2 s. – 50 tys., 1/3 s. – 45 tys., 1/4 s. – 40 tys., 1/8 s. – 30 tys., poniżej 1/8 s. – 200 zł za cm².

Dodatki do ceny podstawowej
– za każdy dodatkowy kolor + 30%,
– za każdy specjalny kolor (nie wynikający z podstawowych kolorów) + 30%,
– za pełny kolor (grafika wielobarwna, zdjęcia kolorowe) + 120%,
– za zamieszczenie ogłoszenia na I lub IV stronie okładki + 100%,
– za zamieszczenie ogłoszenia na II i III stronie okładki + 50%.

Zniżki
dotyczą ogłoszeń – całkowitych powtórzeń
– za ogłoszenia 3–5-krotne – 10%
– za ogłoszenia 6–10-krotne – 20%
– za ogłoszenia 11-krotne i powyżej – 30%
– za artykuły i wkładki reklamowe wykonane przez zleceniodawcę – 40%
– za biuletyny i bloki reklamowe – 60%
W innych uzasadnionych wypadkach dopuszcza się stosowanie rabatów specjalnych.

Ceny nadbitek reklamowych

wkładka 2 s. A4	
– nakład do 500 egz.	30 tys. zł
– za każde następne 500 egz.	25 tys. zł
wkładka 4 s. A4	
– nakład do 500 egz.	60 tys. zł
– za każde następne 500 egz.	50 tys. zł

Ceny usług nie wymienionych w niniejszym cenniku będą ustalane każdorazowo (w oparciu o kalkulację), jako ceny umowne.

W wypadku rezygnacji Zleceniodawcy z wykonania zamówienia przed przekazaniem materiałów do druku – ponosi on koszty w wysokości 25% wartości zlecenia.
W wypadku rezygnacji – gdy materiały są już w druku – Zleceniodawca ponosi pełne koszty ogłoszenia.

Niniejszy cennik dotyczy wyłącznie ogłoszeń firm krajowych i obowiązuje od 1 stycznia 1989 r. Ogłoszenia przyjęte przed tym terminem będą rozliczane według dotychczas obowiązującego cennika.

Ogłoszenia przyjmowane są przez:
Dział Ogłoszeń i Reklamy WCiKT NOT SIGMA
ul. Świętojerska 5/7, 00-236 Warszawa
adres do korespondencji: skrytka pocztowa 1004, 00-950 Warszawa
telefony: 31-93-65 lub 31-22-21 w. 196 i 291

Perspektywy rozwoju sieci przemysłowych

Opracowany przez samochodową firmę General Motors standard sieci lokalnych do zastosowań przemysłowych, znany pod nazwą MAP (ang. *Manufacturing Automation Protocol* – protokół automatyzacji wytwarzania), staje się coraz bardziej popularny w miarę jak sieci tego rodzaju stają się konkurencyjne z firmowymi sieciami dominującymi obecnie na rynku.

Jedną z przyczyn tego stanu rzeczy jest opracowywanie coraz to nowych układów, zwykle w postaci standardowych kostek, spełniających wymagania MAP. Zapowiedziany jest np. modem służący do pracy z falą nośną (ang. *carrier-band*) oznaczony symbolem MC68194 firmy Motorola Inc.

Podobne układy opracowują również inni wytwórcy, np. Siemens AG w RFN opracowuje taki modem o nazwie SAB 82511, a Intel Corp. planuje całą rodzinę układów dla MAP w swym ośrodku Data Communications Component Operation w Folsom, w Kaliforni.

Układy z pasmem fali nośnej zastąpią realizacje na płytkach, stosowane dotychczas w modemach z falą nośną.

Zakłada się, że sieci fabryczne będą oparte na systemie szerokopasmowym, lecz będą dołączone do podsieci pracujących w paśmie fali nośnej, w których wykorzystuje się przełączenie tylko dwóch częstotliwości do przesyłania komunikatów. Układy są wówczas znacznie prostsze od szerokopasmowych, wymagających elementów pracujących z częstotliwościami radiowymi i złożonych układów do przesyłania zmodulowanych sygnałów wieloma kanałami.

Ponadto podsieci z pasmem fali nośnej są bardziej podatne na modyfikacje i bardziej dostosowane do sterowania procesami, w których wielkością krytyczną jest czas. Obecnie bada się 10 pilotowych sieci w różnych zakładach firmy General Motors. Na początku 1986 r. rozpoczęto ograniczone badania systemu z pasmem fali nośnej, a na wiosnę 1987 r. – pilotową produkcję tych podsieci.

Według Venture Development Corp. zapotrzebowanie na sieci MAP z pasmem fali nośnej będzie gwałtownie wzrastać, osiągając ponad 6 tysięcy systemów w 1990 r. w porównaniu z niespełna trzema tysiącami sieci szerokopasmowych; cena za węzeł sieci z pasmem fali nośnej wyniesie 1375 dolarów, a szerokopasmowych – 1860 dolarów. Inne prognozy przewidują jeszcze większy spadek cen sieci z pasmem fali nośnej do 600 dolarów lub mniej.

Obecnie osiągalne modemy kosztują rzędu 700 dolarów, natomiast wspomniana kostka Motoroli, realizowana w technice ECL o szerokości ścieżek 3 μm i obudowie typu PLCC (ang. *Plastic Leaded Chip Carriers* – zob. Elektronika, nr 7-8 1986) z 44 końcówkami, ma kosztować początkowo 30-35 dolarów, a po kilku latach 10-15 dolarów.

Motorola przewiduje też produkcję pakietu zawierającego oprócz modemu i układów pomocniczych także układy sterowania znamieniowo-magistralową siecią MAP (ang. *token bus*). Początkowy koszt takiego pakietu wynosiłby 1000-1200 dolarów, z perspektywą spadku do ok. 500 dolarów po kilku latach.

J.R

Światowy rynek mikroprocesorów 32-bitowych RISC

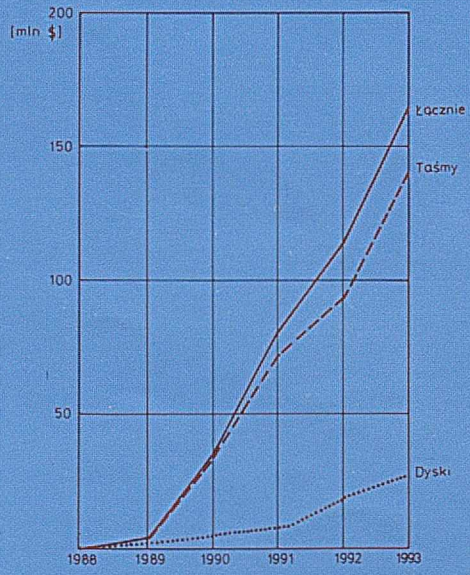
Całkowite wpływy za 32-bitowe kostki typu RISC wzrosną z 17 mln dolarów w 1987 roku do 505 mln w roku 1992, co daje średni roczny wskaźnik wzrostu 96,3%. Analogiczny parametr dla mikroprocesorów 32-bitowych o rozszerzonej liście rozkazów wynosi 24,8%. Udział mikroprocesorów RISC w produkcji mikroprocesorów 32-bitowych wzrośnie z 6,1% w 1987 roku do 38,7% w 1992 roku. Dane te pochodzą z szacunków firmy Information Network. Komputery oparte na zredukowanej liście rozkazów mają wykazywać w tym samym okresie średni współczynnik wzrostu 57,6% w porównaniu z 8,7% dla wszystkich komputerów. Udział komputerów RISC w sprzedaży zwiększy się z 7% w 1987 roku do 44,5% w 1992 roku.

Główne zastosowania procesorów opartych na zredukownej liście rozkazów to produkcja jednostek centralnych komputerów, sterowanie procesami w przemyśle i drukarki laserowe. W tych ostatnich zastosowaniach procesory te są stosowane jako wbudowane sterowniki. Udział mikrokomputerów z takim wykorzystaniem kostek RISC ma wzrosnąć z 0,12% w 1988 roku do 12,4% w 1992 roku. Największy wzrost zastosowań sterowników RISC nastąpi w sterowaniu procesami przemysłowymi, gdzie w analogicznym okresie ma wzrosnąć z 0,18% do 22%. Te same wskaźniki dla drukarek laserowych wynoszą 0,1 i 9,6%, przy czym dla drukarek najszybszych, o szybkości druku powyżej 21 stron na minutę, mają one wartości 0,65 i 33,6%. **(JR)**

Papier cyfrowy – nowy środek przechowywania informacji

W kwietniu 1988 roku wprowadzono po raz pierwszy w Stanach Zjednoczonych nowy nośnik, na którym można przechowywać duże ilości danych po niskich kosztach – zaledwie kilka dolarów za 1 GB. Jest to papier cyfrowy, którego produkcję na pełną skalę miał rozpocząć brytyjski koncern chemiczny ICI w pierwszym kwartale 1989 roku. Natomiast pierwsze urządzenia napędowe wykorzystujące ten nośnik są oczekiwane na rynku w drugiej połowie 1989 roku (Venture Development Corp., Natick w stanie Massachusetts).

Papier cyfrowy jest najnowszym nośnikiem z gatunku pamięci optycznych. Jest to cienka elastyczna folia wytwarzana w dużych rolkach, która może być cięta na części o różnych kształtach i wymiarach. Stacje napędowe papieru cyfrowego wykorzystują wiązki światła laserowego do trwałego zapisu informacji w specjalnym, czułym na promieniowanie podczerwone, barwniku znajdującym się na folii. Nośnik ten doskonale nadaje się do celów archiwalnych, gdyż informacja raz zapisana jest przechowywana przez ponad 15 lat i nie może być zmieniona ani celowo, ani przypadkowo. Badania rynku USA przeprowadzone przez firmę Venture wykazały, że od roku 1993 średni wskaźnik przyrostu produkcji rocznej będzie wynosić 270%, osiągając milion dysków i 45,4 tys. taśm wartości 165 milionów dolarów.



Papier cyfrowy daje największe upakowanie informacji jakie kiedykolwiek zostało osiągnięte w ośrodkach pamięciowych. Na jednej szpuli o długości około 1 km można zapisać tyle informacji co na 5 tysiącach taśm magnetycznych, 1666 dysków kompaktowych pamięci stałych (CD - ROM), tysiącu dysków optycznych z jednokrotnym zapisem (WORM – *write once, read many*), 3 milionach dyskietek elastycznych lub miliardzie arkuszy drukowanego tekstu.

Papier cyfrowy jest również znacznie tańszy. Dysk z papieru cyfrowego o pojemności 1 GB kosztuje 50 dolarów, wobec 200 dolarów dla wymazywalnego dysku optycznego i 500 dolarów dla dysku typu WORM. Taśma z papieru cyfrowego jest dziesięciokrotnie tańsza i kosztuje tylko 5 dolarów za 1 GB. Jest ona tańsza nawet od mikrofilmu, który kosztuje około 8 dolarów za GB.

(JR)