2 1987

informatyka

Przechowywanie wiedzy w bazach danych

Współbieżne systemy przetwarzania obrazów

Programator pamięci UVEPROM

Nauczanie informatyki

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

2 1987

Nr 2
Miesięcznik Rok XXI
Luty 1987

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

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

Zakł. Graf. „Tamka". Zam. 0981-1300/86. Obj. 4,0 ark. druk. Nakład 7700 egz. K-81.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 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:

- Profesor Robert Kowalski o programowaniu logicznym i komputerach nowej generacji
- Piotr Cofta przedstawia języki asemblerowe, ich stan obecny i perspektywy rozwoju
- Andrzej Grzywak i Jerzy Kołodziej o sieciach komputerowych opartych na mikrokomputerach Mera 60
- Stanisław Góral i inni o użytkowaniu Międzyuczelnianej Sieci Komputerowej
- Grzegorz Trawiński o pakiecie zintegrowanym COMPUTEX
- Przemysław Rokita o nowej wersji programu szeregującego wysokiego poziomu dla systemu operacyjnego George 3

WITOLD ŁUKASZEWICZ
Instytut Informatyki
Uniwersytet Warszawski

POLSKIE TOWARZYSTWO INFORMATYCZNE

Pierwsza wersja artykułu została przedstawiona podczas Szkoły PTI „Bazy danych" — 3—6 grudnia 1985 r.

Przechowywanie wiedzy i niewiedzy w bazach danych (I)

Jednym z problemów ogólnej teorii baz danych jest problem przechowywania wiedzy i manipulowania tą wiedzą, rozumianą jako zbiór faktów opisujących pewną dziedzinę zastosowań. Bazy danych, których zawartością jest wiedza, nazywane są bazami wiedzy (ang. knowledge bases). Spośród różnych możliwych sposobów reprezentowania wiedzy w bazach danych, coraz więcej zwolenników ma podejście logiczne zakładające, że wiedza jest reprezentowana jako zbiór formuł pewnego systemu logicznego, na ogół klasycznej logiki pierwszego rzędu.

Celem pierwszej części artykułu jest uzasadnienie konieczności stosowania wnioskowania niemonotonicznego przez systemy sztucznej inteligencji oraz przedstawienie podstawowych problemów związanych z jego formalizacją. W drugiej części zostaną zaprezentowane dwa konkretne formalizmy wnioskowania niemonotonicznego.

LOGICZNA REPREZENTACJA WIEDZY

Zalety logicznego podejścia do problemu reprezentacji wiedzy zilustrujemy przykładem, który jednocześnie pozwoli przypomnieć podstawowe pojęcia klasycznej logiki pierwszego rzędu. Rozważmy następującą bazę wiedzy opisującą pewien mikroświat ptaków:

(1) **ptak(Agata)**
(2) **kanarek(Sylwia)**
(3) **kanarek(Kuba)**
(4) **struś(Maciek)**
(5) **lubi(Kuba,Sylwia)**
(6) $\forall x$**(lubi(Maciek,x))**

Powyższa baza wiedzy jest zapisana w języku klasycznej logiki pierwszego rzędu. **Agata, Sylwia, Kuba** i **Maciek** są symbolami stałych, interpretowanymi jako nazwy konkretnych obiektów występujących w opisywanym świecie. Nazwy **ptak, kanarek** i **struś** są symbolami jednoargumentowych **relacji** (**predykatów**), interpretowanymi jako nazwy konkretnych jednoargumentowych relacji, określonych na obiektach świata. Jednoargumentowe relacje są nazywane własnościami. Formuła (1) stwierdza, że obiekt o nazwie **Agata** posiada własność o nazwie **ptak** lub po prostu, że **Agata jest ptakiem**. Formuły (2)—(4) stwierdzają, że **Sylwia** i **Kuba są kanarkami**, natomiast **Maciek** jest **strusiem**. Symbol **lubi** jest symbolem relacji dwuargumentowej, interpretowanym jako nazwa pewnej konkretnej dwuargumentowej relacji określonej na obiektach świata. Formuła (5) stwierdza, że obiekty o nazwach **Kuba** i **Sylwia** znajdują się w tej relacji, czyli że **Kuba lubi Sylwię**. Należy podkreślić, że nie wynika stąd wcale, że Sylwia lubi Kubę.

Formuły (1)—(5) są formułami atomowymi. Formuły atomowe stwierdzają bardzo proste fakty, a mianowicie, że pewne obiekty opisywanego świata znajdują się w pewnych relacjach. Z formuł atomowych buduje się formuły złożone. Przykładem takiej formuły jest formuła (6). Symbol $\forall$ jest kwantyfikatorem ogólnym, który czytamy **dla każdego**. Symbol x jest symbolem zmiennej. Zmienne reprezentują obiekty świata. Formuła (6) stwierdza, że dla każdego obiektu x obiekt o nazwie **Maciek** znajduje się w relacji o nazwie **lubi** z obiektem x. Mówiąc prościej, że **Maciek wszystkich lubi**.

Konstrukcja logicznej bazy wiedzy opisującej pewien konkretny świat, sprowadza się do podania zbioru formuł reprezentujących fakty zachodzące w tym świecie. Fakty najbardziej oczywiste, opisujące podstawowe związki, nazywa się faktami pierwotnymi, a reprezentujące je formuły — aksjomatami. Wszystkie inne fakty, a dokładniej — reprezentujące je formuły zwane twierdzeniami, wyprowadza się z aksjomatów na drodze logicznego wnioskowania. Zasady tego wnioskowania są określone w postaci zbioru tzw. reguł wnioskowania, które tworzą pewną **strukturę dedukcyjną**. Reguły wnioskowania mają charakter czysto syntaktyczny, tzn. są pewnymi manipulacjami na formułach traktowanych jako napisy. Każda reguła wnioskowania jest zawsze postaci:

$$\frac{A_1, \ldots, A_n}{B}$$

Powyższy zapis należy czytać:

z formuł A_1, ..., A_n można wnioskować formułę B.

Zbiór twierdzeń, które można wyprowadzić z danego zbioru aksjomatów, definiuje się jako najmniejszy zbiór spełniający następujące dwa warunki:

(1) **Każdy aksjomat jest twierdzeniem**

(2) **Jeśli A_1, ..., A_n są twierdzeniami oraz $\frac{A_1, \ldots, A_n}{B}$ jest regułą wnioskowania, to B jest twierdzeniem**

Inaczej mówiąc twierdzeniami są aksjomaty oraz formuły wyprowadzalne z nich w wyniku stosowania reguł wnioskowania.

Różne podręczniki logiki podają różne zestawy reguł wnioskowania dla klasycznej logiki pierwszego rzędu. W celu uzyskania pełnego opisu świata do zestawów tych są dołączane odpowiednie zbiory dodatkowych aksjomatów. Aksjomaty te nie opisują żadnej konkretnej rzeczywistości, ale reprezentują fakty prawdziwe w każdym świecie. Nazywa się je aksjomatami logicznymi. Typowym przykładem formuły, która może być uznana za aksjomat logiczny, jest każda formuła postaci $A \vee \sim A$ stwierdzająca, że pewien fakt zachodzi lub nie zachodzi.

W systemach automatycznego dowodzenia twierdzeń wnioskowanie zwykle nie jest oparte bezpośrednio na re-

Dr WITOLD ŁUKASZEWICZ w 1970 r. ukończył Wydział Matematyki Uniwersytetu Warszawskiego. Od tego roku pracował jako stażysta, asystent, starszy asystent i adiunkt w Instytucie Informatyki Uniwersytetu Warszawskiego. W 1979 r. obronił pracę doktorską związaną z automatycznym dowodzeniem twierdzeń w logikach czasu. Od 1983 r. pracuje jako kierownik Zakładu Systemów Informatycznych i Oprogramowania w Instytucie Informatyki Uniwersytetu Warszawskiego.

gułach klasycznej logiki pierwszego rzędu, lecz na tzw. regule rezolucji. Reguła ta jest formalnie dość skomplikowana. Jedną z jej zalet jest to, że nie wymaga ona dodawania żadnych aksjomatów logicznych.

Należy podkreślić, że w logice klasycznej problem automatycznego dowodzenia twierdzeń jest nierozstrzygalny. Nie istnieje bowiem żaden algorytm, który dla danego zbioru aksjomatów **X** oraz danej formuły **A** rozstrzyga, czy **A** jest twierdzeniem wyprowadzalnym z **X**. Wszystkim, co można osiągnąć i co charakteryzuje rzeczywiste systemy dowodzenia twierdzeń w logice pierwszego rzędu, jest skonstruowanie algorytmu, który dla zadanej formuły wyprowadzalnej z danego zbioru aksjomatów zatrzymuje się i stwierdza, że formuła jest wyprowadzalna. W wypadku przeciwnym algorytm ten może się nigdy nie zatrzymać.

Fakt, że formuła **A** jest wyprowadzalna z danego zbioru formuł **X**, zapisujemy $X \vdash A$.

Niech **X** będzie zbiorem formuł zawartych w pewnej bazie wiedzy. Mówimy, że baza ta jest sprzeczna wtedy i tylko wtedy, gdy dla każdej formuły **A** zachodzi $X \vdash A$ W przeciwnym wypadku bazę nazywamy niesprzeczną. Mówimy, że baza jest zupełna wtedy i tylko wtedy, gdy dla każdej formuły **A** zachodzi $X \vdash A$ **lub** $X \vdash \sim A$. W przeciwnym wypadku mówimy, że baza jest niezupełna.

Wracając do bazy wiedzy opisującej mikroświat ptaków, można wykazać, że jest ona niezupełna, a zatem i niesprzeczna. Posługując się aksjomatami (1)—(6) nie można na przykład stwierdzić, ani że **Sylwia jest ptakiem**, ani że nie jest. Wynika to stąd, że nasza baza nie zawiera informacji mówiącej o tym, że kanarki są ptakami. Aby uwzględnić ten fakt, należałoby dodać nowy aksjomat [1]):

$$\forall x\,(\text{kanarek}(x) \supset \text{ptak}(x))$$

Analizując bazę wiedzy określoną aksjomatami (1)—(6), stwierdzono, że reprezentuje ona pewien świat ptaków. Hipotezę tę uzasadnia jednak wyłącznie semantyka użytych nazw. Nazwy języka logiki mogą reprezentować dowolne obiekty i dowolne relacje zachodzące między nimi. Należy zatem spodziewać się, że ta sama reprezentacja może opisywać fakty prawdziwe w wielu różnych światach. Aby się przekonać, że tak jest w istocie, rozważmy świat liczb naturalnych. Jeśli nazwiemy:

liczbę 1 — Maciek
liczbę 2 — Kuba
liczbę 3 — Sylwia
liczbę 4 — Agata
własność bycia liczbą parzystą — ptak
własność bycia liczą pierwszą — kanarek
własność bycia liczbą mniejszą niż 5 — struś
relację $\leqslant$ — lubi

to czytelnik może się przekonać, że wszystkie fakty reprezentowane przez aksjomaty (1)—(6) są prawdziwe w świecie liczb naturalnych. Przykładowo, aksjomat (6) stwierdza, że liczba 1 jest równa bądź mniejsza od dowolnej liczby naturalnej. Oczywiście nikt rozsądny nie użyje tego rodzaju nazw do opisu świata liczb naturalnych. Nie zmienia to jednak faktu, że skoro obiekty i relacje opisywanego świata można nazwać dowolnie, to aksjomaty (1)—(6) należy uznać za formalnie poprawną reprezentację fragmentu wiedzy o liczbach naturalnych.

Fakt, że baza wiedzy może równocześnie reprezentować różne, często bardzo odległe dziedziny, nie prowadzi do żadnych istotnych komplikacji. Logika posługuje się czysto syntaktycznym mechanizmem wnioskowania, sprowadzającym się do manipulowania formułami jako napisami. Reguły wnioskowania używane w systemach logicznych nie są przypadkowe. Wybiera się tylko takie reguły, w których prawdziwość przesłanek gwarantuje prawdziwość konkluzji, bez względu na specyficzne cechy reprezentowanego świata. Zauważmy, że każdy proces logicznego wnioskowania rozpoczyna się od aksjomatów. A zatem bez względu na własności opisywanego świata, prawdziwość aksjomatów zapewnia prawdziwość wyprowadzalnych z nich twierdzeń. Oczywiście, w różnych światach twierdzenia te mają inną interpretację. Tym niemniej zawsze reprezentują prawdziwe fakty.

[1]) Znak $\supset$ oznacza implikację (przyp. red.)

Powyższą uwagę zilustrujemy na przykładzie bazy wiedzy zadanej aksjomatami (1)—(6). Można wykazać, że z aksjomatu (6) wynika formuła:

lubi (Maciek,Agata)

W świecie ptaków formuła ta reprezentuje fakt, że **Maciek lubi Agatę**. Wynika to natychmiast z faktu, reprezentowanego aksjomatem (6), że **Maciek lubi wszystkich**. W świecie liczb naturalnych powyższa formuła opisuje fakt, że **liczba naturalna 1 jest nie większa od liczby 4**. Tym razem wynika to z faktu, również reprezentowanego aksjomatem (6), że **liczba 1 jest najmniejszą liczbą naturalną**.

Powyższa dyskusja pozwala sformułować następujące zalety logicznej reprezentacji wiedzy w bazach danych:

(1) dobrze określona struktura dedukcyjna systemów logicznych, umożliwiająca sprowadzenie wnioskowania w opisywanym świecie do dowodzenia twierdzeń w logice formalnej;

(2) dobrze określona, jasna i ogólnie akceptowana (przynajmniej w wypadku klasycznej logiki pierwszego rzędu) semantyka, pozwalająca traktować zawartość bazy jako opis pewnej rzeczywistości, a nie jako zbiór abstrakcyjnych napisów;

(3) proste i jednoznaczne konwencje notacyjne wpływające na czytelność;

(4) precyzyjnie określone pojęcia, takie jak niesprzeczność czy zupełność, trudne do zdefiniowania w innych metodach reprezentacji wiedzy.

Logiczna baza wiedzy jest zbiorem formuł pewnego systemu logicznego. W literaturze można znaleźć bardzo wiele różnych systemów logicznych. Większość z nich posiada strukturę dedukcyjną opartą na założeniu, że każda reguła wnioskowania ma postać:

z formuł A_1, ..., A_n wnioskuj formułę B

Formuły A_1, ..., A_n są aksjomatami albo twierdzeniami, zatem każdy wniosek, który można w tych systemach wyprowadzić, wynika wyłącznie z aksjomatów i wcześniej udowodnionych twierdzeń. Systemy logiczne o powyższej strukturze dedukcyjnej będziemy nazywać systemami standardowymi. Można więc powiedzieć, że konkluzje, które wyprowadzamy w systemach standardowych, są oparte na naszej wiedzy.

Niech **X** będzie zbiorem formuł pewnego systemu logicznego. Zbiór wszystkich twierdzeń wyprowadzalnych ze zbioru **X** oznaczamy przez **Th (X)**. Formalnie:

$$\text{Th}(X) = \{A\colon X \vdash A\}$$

Specyficzną cechą wszystkich standardowych systemów logicznych jest ich monotoniczność. Własność ta określa, że każde twierdzenie wyprowadzalne z danego zbioru aksjomatów pozostaje twierdzeniem, jeśli do zbioru aksjomatów dodamy nową formułę. Inaczej mówiąc, zbiór twierdzeń zwiększa się monotonicznie wraz ze zwiększeniem liczby aksjomatów:

$$\text{jeśli } X \subseteq Y, \text{ to } \text{Th}(X) \subseteq \text{Th}(Y)$$

Istnieją zastosowania, szczególnie w dziedzinie sztucznej inteligencji, wymagające baz wiedzy opartych na niestandardowych systemach logicznych. Specyficzną cechą tych systemów jest ich niemonotoniczność, tzn. możliwość wyprowadzenia twierdzenia, które może zostać unieważnione w wyniku dodania nowego aksjomatu. Bierze się to z faktu, że konkluzje wyprowadzalne w niestandardowych systemach logicznych zależą nie tylko od wiedzy, ale również od niewiedzy [2]). Można zatem powiedzieć, że bazy wiedzy oparte na niestandardowych systemach logicznych pośrednio zawierają także niewiedzę.

PROBLEMY WNIOSKOWANIA NIEMONOTONICZNEGO

Sztuczna inteligencja zajmuje się konstruowaniem systemów symulujących inteligentne zachowanie istot ludzkich. Jest zatem oczywiste, że formalizacja mechanizmów wnioskowania specyficznych dla ludzi jest centralnym

[2]) Od tego, czego nie wiemy — w literaturze polskiej używa się też terminu „ignorancja" (przyp. red.)

problemem tej dziedziny. Sposób wnioskowania charakterystyczny dla ludzi nosi nazwę wnioskowania naturalnego (ang. common-sense reasoning).

Specyficzną cechą wnioskowania naturalnego jest konieczność uwzględniania niewiedzy w procesie wyprowadzania konkluzji. Zilustrujemy to następującym przykładem [5].

Zakładam, że planuję podróż samochodem, który zostawiłem wczoraj na parkingu niewidocznym z okien mojego mieszkania. Jeżeli jestem człowiekiem racjonalnym, to po samochód udam się na parking. Oznacza to, że właśnie tam spodziewam się go znaleźć. Z drugiej strony, ponieważ parking jest niewidoczny z okien mojego mieszkania, nie może wynikać z mojej wiedzy konkluzja, że samochód znajduje się na parkingu. W rzeczywistości potrafię sobie wyobrazić wiele różnych sytuacji, w których samochodu na parkingu nie ma. Na przykład, ktoś mógł go ukraść. Jeżeli jednak koniecznie chciałbym uwzględnić wszystkie okoliczności, które mogą uniemożliwić wykonanie zamiaru, to nigdy nie byłbym w stanie zrealizować żadnej czynności. Zamiast zatem siedzieć i analizować wszystkie możliwe scenariusze, akceptuję regułę wnioskowania:

jeśli nie wiesz, że jest inaczej, załóż, że samochód jest tam, gdzie go zostawiłeś

i udaję się na parking.

Należy podkreślić, że każda konkluzja wyprowadzona na podstawie reguły uwzględniającej niewiedzę może zostać unieważniona, ponieważ nowa informacja zmniejsza naszą niewiedzę. Chcąc rozpocząć podróż zakładam, że samochód jest na parkingu. Ale informacja, że samochód został skradziony, natychmiast obala powyższe założenie.

W codziennym życiu ciągle akceptujemy konkluzje, które mogą być unieważnione w wyniku dopływu nowych informacji. Tego rodzaju wnioski nazywane są **przekonaniami** (ang. belief). Umiejętność ich wyprowadzania sprawia, że wnioskowanie naturalne jest niemonotoniczne, tzn. zbiór konkluzji nie zwiększa monotonicznie wraz ze wzrostem zbioru przesłanek.

Aby zilustrować niektóre problemy związane z niemonotonicznością, rozważmy regułę:

jeśli pewna osoba jest dzieckiem, to jeśli nie wiesz, że jest inaczej, załóż, że osoba ta ma rodziców

Wiarygodność powyższej reguły opiera się na obserwacji, że dzieci na ogół mają rodziców. Jest to oczywiście reguła niemonotoniczna. Jeśli cała nasza wiedza o Janku sprowadza się do faktu, że jest on dzieckiem, to reguła ta pozwala nam założyć, że **Janek ma rodziców**. Założenie to musimy jednak natychmiast odrzucić, jeśli dowiemy się, że **Janek jest sierotą**.

Dwie przybliżone reprezentacje powyższej reguły w standardowej logice pierwszego rzędu brzmią następująco:

$$(1)\forall x(\text{dziecko}(x) \supset \text{ma-rodziców}(x))$$

$$(2)\forall x(\text{dziecko}(x) \wedge \sim \text{sierota}(x) \supset \text{ma-rodziców}(x))$$

Pierwsza z powyższych reprezentacji jest za silna. Opisuje ona fałszywy fakt, że wszystkie dzieci mają rodziców. Umieszczenie (1) w bazie wiedzy jest ryzykowne. Może bowiem łatwo doprowadzić do sprzeczności bazy. Druga z powyższych reprezentacji jest za słaba. Posługując się nią, nie można założyć o typowym dziecku, że ma ono rodziców. Założenie to wymaga dodatkowej informacji, że rozważane dziecko nie jest sierotą.

To, czego naprawdę potrzeba, jest czymś pośrednim między reprezentacjami (1) i (2). Potrzebny jest pewien mechanizm umożliwiający wnioskowanie, że typowe dziecko ma rodziców, ale równocześnie blokujący ten wniosek, jeśli prowadzi on do sprzeczności.

Konstrukcja systemu niemonotonicznego zakłada rozwiązanie dwóch różnych problemów. Można je nazwać problemem formalizacji i problemem weryfikacji przekonań. Pierwszy z nich można sformułować następująco: określić sposób reprezentacji faktów opisywanego świata oraz zdefiniować zbiór przekonań wyprowadzalnych z tej reprezentacji. Problem weryfikacji przekonań jest problemem reorganizacji dotychczas wyprowadzonych przekonań w wypadku, kiedy niektóre z nich zostają obalone w wyniku uzupełnienia bazy wiedzy o nowy fakt. Wbrew pozorom jest to problem bardzo skomplikowany. Trudność polega na tym, że nie wystarczy usunięcie tylko tych przekonań, które są sprzeczne z nowo wprowadzoną informacją. Należy zdawać sobie sprawę, że każde przekonanie może być użyte przy wyprowadzaniu innych. Usuwając zatem pewne przekonanie, należy jednocześnie usunąć wszystkie jego konsekwencje. Problem weryfikacji przekonań nie będzie omawiany w niniejszym artykule — zainteresowanego czytelnika odsyłamy do prac [1—4].

Wiele wczesnych systemów konstruowanych na użytek sztucznej inteligencji posiadało wbudowane mechanizmy niemonotoniczne (wiele przykładów można znaleźć w [5]). Okazało się jednak, że te ad hoc budowane narzędzia działały poprawnie tylko w najprostszych przypadkach. Nie posiadały bowiem żadnych podstaw teoretycznych, więc ich działanie w bardziej skomplikowanych sytuacjach było całkowicie niejasne. Stało się oczywiste, że problem niemonotoniczności wymaga bardziej formalnego podejścia.

LITERATURA

[1] Doyle J.: A Truth Maintenance System. Artificial Intelligence, No. 42. pp. 231—272, 1979

[2] Doyle J.: Some Theories of Reasoned Assumptions — An Essay in Rational Psychology. Carnegie-Mellon University Report, 1982

[3] Martins J., Shapiro S.: A Model for Beliefs Revision. AAAI Workshop on Non-Monotonic Reasoning, New York, pp. 241—294, 1984

[4] McAllester D.: An Outlook in Truth Maintenance, Report 551, Massachusetts Institute of Technology, Artificial Intelligence Lab, 1980

[5] Winograd T.: Extended Interface Models in Reasoning by Computer Systems. Artificial Intelligence, No. 13, pp. 5—26, 1980.

Konferencje

Wielkie Bazy Danych (VLDB)

W dniach od 1 do 4 września 1987 r. odbędzie się w Brighton (W. Brytania) 13. Międzynarodowa Konferencja nt. Wielkich Baz Danych (13th International Conference on Very Large Data Bases).

Program konferencji obejmować będzie następującą tematykę:

- modele danych,
- metody i narzędzia projektowania,
- rozproszone bazy danych,
- optymalizacja zapytań,
- sterowanie współbieżnością,
- maszyny baz danych,
- zagadnienia wydajności,
- metody zabezpieczania,
- bazy wiedzy,
- bazy danych z zastosowaniem wielu mediów,
- metody wdrażania,
- modele ukierunkowane na obiekty,
- rola logiki.

Bliższych informacji udziela: Miss Christine Edginton, Conference Manager, BISL Conference Department, The British Computer Society, 13 Mansfield Street, London WiM OBP, tel. 01-637-0471.

DANIEL TABAK
Electrical and Computer Engineering Department
George Mason University
Fairfax, stan Virginia
USA

Intel 80386 i nowe mikroprocesory 32-bitowe (2)

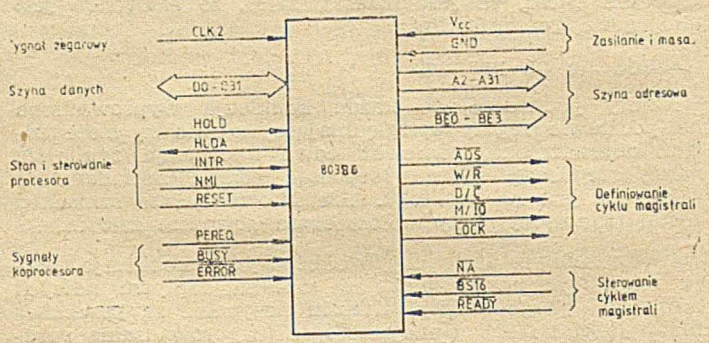
W drugiej części artykułu omówiono organizację mikroprocesora 80386 oraz porównano go z innymi mikroprocesorami 32-bitowymi: MC 68020, NS 32332, Z80000 i WE 32100.

ORGANIZACJA MIKROPROCESORA 80386

Mikroprocesor 80386 zrealizowano w technologii CHMOS III, o szerokości ścieżek 1,5 μm. Jego maksymalna częstotliwość zegarowa wynosi 16 MHz. Mikroukład zawiera ponad 275 000 tranzystorów i jest umieszczony w 132-końcówkowej obudowie. Rozmieszczenie końcówek mikroukładu przedstawiono na rys. 1. Jak widać na rysunku, 32-bitowa szyna danych i szyna adresowa są rozdzielone, niemultipleksowane. Dane są przesyłane po liniach D0-D31, a adresy po liniach A2-A31 i BE0-BE3 (linie A0 i A1 są wewnętrzne). Wyjścia BE (ang. byte enable), aktywne przy niskim poziomie sygnału określają, który bajt szyny danych jest związany z bieżącym przesyłaniem informacji:

BE0 odpowiada bajtowi D0-D7
BE1 — bajtowi D8-D15
BE2 — bajtowi D16-D23
BE3 — bajtowi D24-D31

Schemat blokowy mikroprocesora 80386 przedstawiono na rys. 2. Mikroprocesor składa się z sześciu jednostek logicznych działających potokowo (ang. pipelined). Każda jednostka wykonuje określone zadanie lub jeden krok podczas pobierania i wykonywania rozkazu:



Rys. 1. Rozmieszczenie końcówek mikroprocesora 80386 (rysunki reprodukowano za zgodą Intel Corp.)

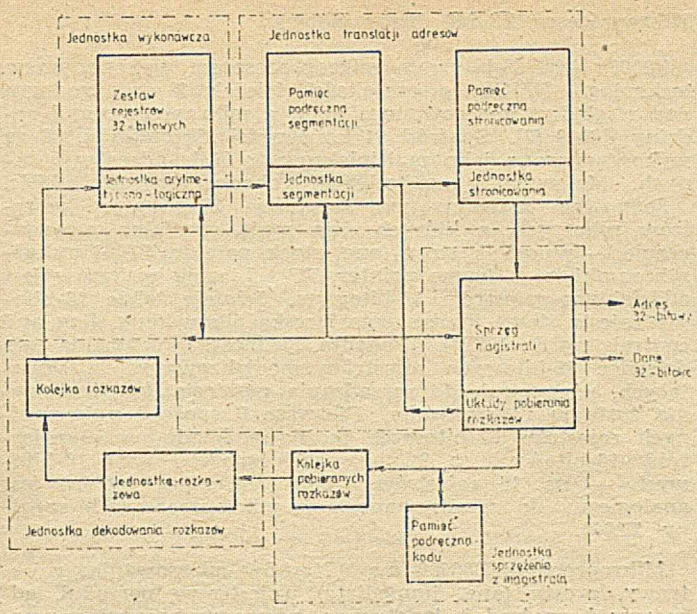
- jednostka sprzężenia z magistralą, **BIU** (ang. bus interface unit), sprzęga jednostkę centralną z zewnętrzną magistralą systemową i steruje przepływem sygnałów danych, adresowych i sterujących,
- jednostka pobierania wstępnego, **PRU** (ang. prefetch unit), jest odpowiedzialna za pobieranie rozkazów z pamięci; zawiera 16-bajtową kolejkę,
- jednostka dekodowania rozkazów, **IDU** (ang. instruction decode unit), dekoduje i przygotowuje rozkazy do bezpośredniego wykonania; zawiera trójpoziomową kolejkę zdekodowanych rozkazów,
- jednostka wykonawcza, **EU** (ang. execution unit), operuje zdekodowanymi rozkazami, podejmując kroki niezbędne do ich wykonania; zawiera logikę sterującą i uniwersalny zestaw rejestrów 32-bitowych,
- jednostka segmentacji, **SU** (ang. segmentation unit), określa tzw. adres liniowy,
- jednostka stronicowania, **PAU** (ang. paging unit), przekształca adres liniowy na adres fizyczny i kontroluje naruszanie stron.

Mikroprocesor 80386 jest sterowany mikroprogramem, co powoduje zwłokę dwóch cykli zegarowych. Wprowadzając zakładkowanie (ang. overlap) pobierania i wykonywania oraz stosując tzw. opóźniony mikroskok (ang. delayed microjump) udało się skrócić czas wykonywania mikrorozkazu do pojedynczego mikrocyklu, tj. 62,5 ns przy częstotliwości zegarowej 16 MHz.

W mikroukładzie 80386 znajduje się również podsystem zarządzania pamięcią, zapewniający zarządzenie pamięcią wirtualną, opcjonalne stronicowanie i cztery poziomy ochrony kodu. Ponadto mikroukład ma możliwości uruchamiania i samotestowania sprzętu. Specjalny tyb pracy Virtual 8086 Mode umożliwia wykonywanie na tym mikroprocesorze, z ochroną kodu i stronicowaniem, oprogramowania mikroprocesora 8086.

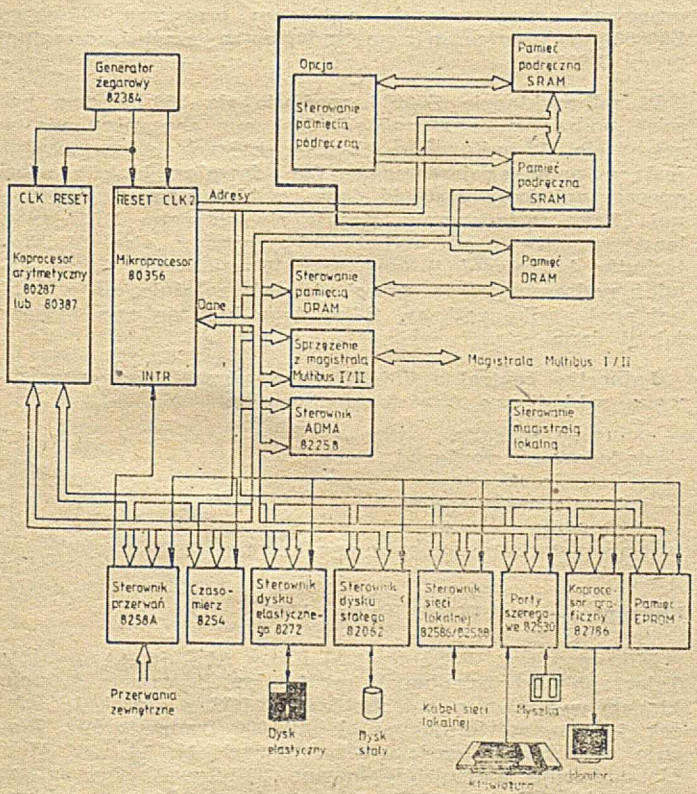
Profesor DANIEL TABAK urodził się w Polsce w 1934 r. Studia ukończył w Technion University (Haifa, Izrael), a doktorat uzyskał na uniwersytecie stanu Illinois (Urbana, USA). Pracował kolejno w General Electric Company, Wolf R and D Corporation, Ronsselaer Polytechnical Institute i Hartford Graduate Center. W 1972 r. został profesorem w Ben Gurion University of the Neger (Beer-Sheva, Izrael), gdzie w latach 1976—1977 i 1979—1983 kierował wydziałem Electrical and Computer Engineering Dept. W latach 1977—1981 przebywał kolejno w NASA Langely Research Center (Hampton, stan Virginia), na uniwersytecie stanu Teksas (Austin, USA) oraz na Uniwersytecie Kalifornijskim (Berkeley, USA).

Jest współautorem (wraz z B. C. Kuo) książki „Optimal Control by Mathematical Programming" (Prentice-Hall, 1971; tłumaczenie rosyjskie: Nauka, 1975). Obecne zainteresowania prof. Tabaka dotyczą architektury komputerów, a także mikrokomputerów i bezpośredniego sterowania cyfrowego. Od września 1983 roku prof. Tabak przebywa w USA, początkowo w Boston University, obecnie w George Mason University.



Rys. 2. Budowa wewnętrzna mikroprocesora Intel 80386

Mikroprocesor 80386 może współpracować z koprocesorem arytmetycznym 80287 lub 80387 i z wieloma starszymi mikroukładami pamięciowymi i sprzęgającymi. Dodatkowo można do niego dołączyć pamięć podręczną (ang. cache memory). Typową konstrukcję mikrokomputera opartego na mikroprocesorze 80386, przedstawiono na rys. 3. Nowym mikroukładem, zaprojektowanym specjalnie do współpracy z tym mikroprocesorem jest koprocesor 82258 ADMA (ang. advanced direct meneory access). Może on obsługiwać jednocześnie 35 różnych urządzeń, jak: terminale, drukarki, dyski itp. W oparciu o mikroprocesor 80386 opracowano dwa nowe komputery jednopłytkowe: iSBC 386/20 i iSBC 386/100, przeznaczone odpowiednio dla systemów Multibus I i Multibus II.



Rys. 3. Przykładowy komputer oparty na mikroprocesorze 80386

INNE MIKROPROCESORY 32-BITOWE

Poniżej dokonano krótkiego przeglądu nowszych mikroprocesorów 32-bitowych: MC68020, NS32332, Z80000 i WE 32100.

Mikroprocesor MC68020 firmy Motorola

Mikroukład MC68020 jest 32-bitowym mikroprocesorem wykonanym w technologii HCMOS, złożonym z 200 000 tranzystorów, umieszczonym w obudowie 120-końcówkowej i pracującym z częstotliwością zegarową 16,67 MHz (tj. w cyklu 60 ns). Szczególnie ważne okazało się zapewnienie zgodności kodu wynikowego mikroprocesora 68000 z pozostałymi mikroprocesorami tej rodziny, tj. MC68000, 68008, 68010 i 68012. Wszystkie wymienione mikroprocesory mają wspólny zestaw rejestrów.

- osiem 32-bitowych rejestrów danych, D0-D7
- siedem 32-bitowych rejstrów adresowych, A0-A6
- 32-bitowy wskaźnik stosu użytkownika, A7
- 32-bitowy licznik rozkazów, PC
- 8-bitowy rejstr kodów warunków, CCR, będący dolnym bajtem 16-bitowego rejestru stanu.

Wszystkie mikroprocesory mają także 32-bitowy wskaźnik stosu trybu Supervisor, A7-1, a mikroprocesory 68010, 68012 i 68020 — także inne rejestry, których można używać tylko w trybie Supervisor.

Mimo ścisłej zgodności z pozostałymi modelami tej rodziny, mikroprocesor 68020 ma znacznie rozszerzoną architekturę. Czternaście trybów adresowania mikroprocesora 68010, w 68020 rozszerzono do osiemnastu następujących [5]:

- bezpośredni przez rejestr danych,
- bezpośredni przez rejestr adresowy,
- pośredni przez rejestr adresowy,
- pośredni przez rejestr adresowy z postinkrementacją,
- pośredni przez rejestr adresowy z predekrementacją,
- pośredni przez rejestr adresowy z przemieszczeniem,
- pośredni przez rejestr adresowy z indekşowaniem (przemieszczenie 8-bitowe),
- pośredni przez rejestr adresowy z indeksowaniem (przemieszczenie bazowe),
- pośredni przez pamięć z postindeksowaniem,
- pośredni przez pamięć z preindeksowaniem,
- pośredni przez licznik rozkazów z przemieszczeniem,
- pośredni przez licznik rozkazów z indeksowaniem (przemieszczenie 8-bitowe),
- pośredni przez licznik rozkazów z indeksowaniem (przemieszczenie bazowe),
- pośredni przez pamięć i licznik rozkazów z postindeksowaniem,
- pośredni przez pamięć i licznik rozkazów z preindeksowaniem,
- bezwzględny krótki,
- bezwzględny długi,
- prosty (ang. immediate).

Sześć pierwszych trybów z indeksowaniem umożliwia również skalowanie, tj. mnożenie indeksu przez współczynnik skali (ang. scale factor) równy 1, 2, 4 lub 8, co po raz pierwszy zastosowano w modelu 68020.

Lista rozkazów, zawierająca 58 rodzajów rozkazów dla mikroprocesora 68010, została znacznie rozszerzona. Oprócz siedmiu nowych rozkazów dla koprocesora MC68881, dodano 16 nowych rozkazów jednostki centralnej [4], tj.:

BFCHG — zmiana pola bitowego (ang. change bit field),
BFCLR — skasowanie pola bitowego (ang. bit field clear),
BFEXTS — wyodrębnienie pola bitowego ze znakiem (ang. bit field signed extract),
BFEXTU — wyodrębnienie pola bitowego bez znaku (ang. bit field unsigned extract),
BFFFO — odszukiwanie pierwszego bitu ustawionego (ang. bit field find first one set),
BFINS — wstawianie pola bitowego (ang. bit field insert),
BFSET — ustawianie pola bitowego (ang. bit field set),
BFTST — testowanie pola bitowego (ang. bit field test),
CALLM — wywołanie modułu (ang. call module),
CAS — porównanie i zamiana (ang. compare and swap),
CAS2 — porównanie i zamiana dwuargumentowa (ang. compare and swap 2-argument),
CHK2 — kontrola zawartości rejestru ze względu na dolne i górne ograniczenie,
CMP2 — porównywanie zawartości rejestru z dolnym i górnym ograniczeniem,
PACK — upakowanie kodu BCD,
RTM — powrót z modułu,
UNPK — rozpakowanie kodu BCD.

W sumie mikroprocesor MC68020 ma 98 własnych rozkazów i 7 rozkazów koprocesora.

Niektóre rozkazy dostępne w poprzednich modelach dostosowano do operowania słowami 32-bitowymi. Na przykład mnożenie dwóch liczb 32-bitowych ma w mikroprocesorze 68020 wynik 64-bitowy, a dzielenie liczby 64-bitowej przez 32-bitową ma 32-bitowy iloraz i 32-bitową resztę. Przemieszczenia rozgałęzień BCC, BRA, BSR i LINK (ang. branch displacements) rozszerzono do 32 bitów.

Spośród 120 końcówek mikroprocesora wykorzystanych jest tylko 114. Szyna danych D0-D31 jest oddzielona od szyny adresowej A0-A31 (niemultipleksowana). Mikroukład ma wyodrębnioną pamięć o pojemności trzech słów do wstępnego pobierania rozkazów i 256-bajtową podręczną pamięć rozkazów. Organizacja mikroprocesora cechuje się znacznym stopniem równoległości. W czasie pobierania rozkazów z pamięci podręcznej mikroukładu można jednocześnie pobierać dane z zewnętrznej pamięci operacyjnej. Adresy danych i rozkazów są obliczane oddzielnie, równolegle i przesyłane oddzielnymi 32-bitowymi ścieżkami wewnętrznymi mikroukładu. Niedawno wprowadzono nową wersję mikroprocesora MC68020, działającą z szybkością 20 MHz.

Zakładkowanie rozkazów (ang. instruction pipelining) jest realizowane trójstopniowo:

— dekodowanie,
— generowanie sterowania,
— wykonanie.

W mikroukładzie 68020 zawarte są trzy oddzielne sumatory 32-bitowe, przeznaczone do:

— obliczania adresów rozkazów,
— obliczania adresów argumentów,
— przetwarzania danych.

Niekiedy ich działanie może być połączone, np. przy mnożeniu dwóch liczb 32-bitowych stosuje się drugi i trzeci z wymienionych sumatorów.

Mikroprocesor NS32332 firmy National Semiconductor

Mikroprocesor 32-bitowy NS32332 jest najnowszym mikroprocesorem rodziny 32000 [6, 8]. Jest w pełni kompatybilny pod względem kodu wynikowego z poprzednimi mikroprocesorami NS32016 i NS32032. Jednakże zawiera szereg istotnych rozszerzeń w porównaniu z mikroprocesorem 32032.

Jednym z głównych rozszerzeń jest 4 gigabajtowa (2^{32} bajtów) wirtualna, jednolita (tzn. niesegmentowana, ang. uniform) przestrzeń adresowa (mikroprocesor 32032 ma przestrzeń o pojemności 16 MB). Zewnętrzny układ zarządzania pamięcią, NS32382, zapewnia tzw. stronicowanie na żądanie (ang. demand paging) — na strony po 512 bajtów. Mikroukład jest zrealizowany w technologii NMOS, o szerokości ścieżek 2,8 μm, ma 84 końcówki, z których 32 stanowią multipleksowane linie adresów i danych AD0-AD31. Obecnie maksymalna częstotliwość pracy jest równa 15 MHz. Ogólna moc obliczeniowa tego mikroprocesora jest ok. trzykrotnie większa niż jego poprzednika 32032.

Do innych ulepszeń w porównaniu z mikroprocesorem 32032 należą:

— 20-bajtowa kolejka wstępnego pobierania rozkazów (8 bajtów w procesorze 32032),

— 32-bitowy rejestr przesuwny (ang. barrel shifter), sumator adresowy i rejestr adresowy (niedostępne w procesorze 32032).

Mikroprocesor 32332 ma 9 trybów adresowania, identycznych jak w procesorze 32032, a także — te same rejestry jednostki centralnej, włącznie z ośmioma 32-bitowymi rejestrami roboczymi R0-R7. Do współpracy z nimi zaprojektowano kilka mikroukładów pomocniczych:

NS 32C382 (32382) — jednostka zarządzania pamięcią,

NS 32C381 (32381) — jednostka zmiennoprzecinkowa,

NS 32301 — jednostka sterowania taktowaniem,

NS 32310 — sprzęg do 64-bitowego koprocesora zmiennoprzecinkowego Weitek WTL 1164/1165. Obecnie istnieje już przeznaczony dla tego mikroprocesora kompilator skrośny języka C i asembler skrośny, działające na komputerze VAX pod nadzorem systemu operacyjnego VMS.

Mikroprocesor Z80000 firmy Zilog

Z80000 jest 32-bitowym mikroprocesorem wykonanym w technologii NMOS, o szerokości ścieżek 2 μm, zgodnym pod względem generowanego kodu z mikroprocesorami rodziny Z8000 [7]. Spośród 64 końcówek mikroukładu, 32 są wykorzystywane jako multipleksowane linie adresów i danych. Maksymalna częstotliwość robocza wynosi 25 MHz.

Jednostka centralna ma szesnaście 32-bitowych rejestrów roboczych, z których dwa mają specjalne przeznaczenie — wskaźnik stosu (ang. stock pointer, SP) i wskaźnik ramki (ang. frame pointer, FP). Licznik rozkazów jest oddzielnym rejestrem 32-bitowym. Istnieje także 16-bitowy rejestr słowa kontrolnego i wskaźników (ang. flag and control word, FCW). Ponadto, jednostka centralna ma dziewięć 32-bitowych rejestrów specjalizowanych i sterujących, używanych do zarządzania pamięcią, konfigurowania systemu i sterowania. Pierwsze 8 spośród uniwersalnych rejestrów 32-bitowych można podzielić na 16 rejestrów 16-bitowych, a 8 pierwszych spośród tych 16-bitowych — na 16 rejestrów 8-bitowych. Ten podział jest analogiczny do zrealizowanego w mikroprocesorach rodziny Z8000.

Mikroprocesor Z80000 ma 9 trybów adresowania: rejestrowy, prosty (ang. immediate), rejestrowy pośredni, adresowy bezpośredni, indeksowy, adresowy bazowy z przemieszczeniem, indeksowy bazowy z przemieszczeniem, względny i indeksowy względny. Jest to o 1 tryb więcej w porównaniu z mikroprocesorami rodziny Z8000.

Podobnie jak mikroprocesor 80386, Z80000 ma wbudowaną jednostkę zarządzania pamięcią. Umożliwia ona stosowanie stronicowanej pamięci wirtualnej o rozmiarze strony równym 1 KB. Możliwe jest zarówno adresowanie liniowe, jak i segmentowe, jako opcje wybierane programowo. Rozmiar segmentu może wynosić 64 KB lub 16 MB, a cała bezpośrednio adresowalna przestrzeń pamięci ma rozmiar 4 GB.

Mikroprocesor Z80000 ma zbiór specjalnych rozkazów do przesyłania danych i poleceń w tzw. rozszerzonej konfiguracji EPA (ang. extended processing architecture), umożliwiającej użycie koprocesorów. Koprocesor arytmetyczny Z8070 (ang. arithmetic processing unit, APU) może pracować asynchronicznie względem procesora głównego Z80000, wykonując własne zadania w czasie normalnej pracy Z80000.

W mikroukładzie Z80000 znajduje się 256-bajtowa pamięć podręczna (ang. cache). Jest ona zorganizowana w szesnaście 16-bajtowych bloków, których odwzorowanie jest w pełni skojarzeniowe, tzn. dowolny blok pamięci operacyjnej może zastąpić dowolny blok pamięci podręcznej. Przy zastępowaniu stosuje się algorytm LRU (ang. least recently used, najrzadziej używany element oraz technikę kalkowania [1]) (ang. write-through technique). Pamięć podręczna może przechowywać, zależnie od wybranej opcji, tylko rozkazy, tylko dane lub zarówno rozkazy jak i dane.

Jednostka centralna Z80000 ma przetwarzanie zorganizowane zakładkowo (sześciostopniowo):

— pobranie rozkazu,
— dekodowanie rozkazu,
— obliczenie adresu,
— pobranie argumentów,
— wykonanie rozkazu,
— zapamiętanie argumentu wynikowego.

Mikroprocesor WE32100 firmy ATT

Mikroprocesor WE32100 [1-3] firmy ATT (ang. American Telegraph and Telephone) jest 132-końcówkowym mikroukładem wykonanym w technologii CMOS, o szerokości ścieżek 1,5 μm, zawierającym ok. 180 000 tranzystorów. Pracuje przy częstotliwościach zegarowych 10, 14 i 18 MHz. Ma oddzielne, niemultipleksowane, 32-bitowe szyny danych i adresów. Bezpośrednio adresowalna przestrzeń adresowa pamięci wynosi 4 GB. Zapewniona jest pamięć wirtu-

[1]) Algorytm LRU stosuje się w wypadku, gdy w pamięci podręcznej nie ma bloku o żądanej zawartości, natomiast technikę kalkowania — polegającą na jednoczesnym uaktualnieniu zawartości pamięci operacyjnej i podręcznej — przy zapisie argumentów (przyp. red.).

alna, stronicowana na żądanie, o rozmiarze strony równym 2 KB. Cała architektura jest zoptymalizowana, ze względu na użycie systemu operacyjnego Unix. Zrealizowano także wejście-wyjście odwzorowane w pamięci.

Jednostka centralna zawiera szesnaście 32-bitowych rejestrów, z których dziewięć, r0-r8, jest uniwersalnych, a pozostałe siedem specjalizowanych:

- wskaźnik ramki **FP** (ang. frame pointer),
- wskaźnik argumentu **AP** (ang. argument pointer),
- słowo stanu procesora **PSW** (ang. processor status ward),
- wskaźnik stosu **SP** (ang. stack pointer),
- wskaźnik bloku kontrolnego procesu **PCBP** (ang. process control block pointer),
- wskaźnik stosu przerwania **ISP** (ang. interrupt stack pointer),
- licznik rozkazów **PC** (ang. program counter).

Ponadto istnieje łącznie 18 trybów adresowania.

Jednostka centralna 32100 ma 5 dodatkowych układów pomocniczych:

- jednostkę zarządzania pamięcią, WE32101,
- sterownik pamięci dynamicznej RAM, WE32103,
- sterownik DMA, WE32104,
- systemową jednostkę sprzęgającą, WE32105,
- jednostkę operacji arytmetycznych wykonującą wszystkie operacje zmiennoprzecinkowe, WE32106.

Mikroprocesor 32100 ma także 256-bajtową pamięć podręczną rozkazów i 8-bajtową kolejkę rozkazów.

PORÓWNANIE MIKROPROCESORÓW 32-BITOWYCH

Niektóre mikroprocesory opisane w poprzednich punktach, np. 80386, NS32332 i WE32100, wprowadzono dopiero w 1985 r., a mikroprocesor Z80000, choć zaanonsowany już w 1983 r., do tej pory nie jest dostępny. Dlatego nie można jeszcze podać ostatecznych wyników ich porównania. Wstępne oceny opublikowane przez firmę Intel dla mikroprocesorów 80386 i MC68020 przedstawiono w tabeli 1.

Cykle odczytu i zapisu czterech mikroprocesorów porównano w tabeli 2, a w tabeli 3 przedstawiono porównanie ogólnych właściwości pięciu omówionych mikroprocesorów 32-bitowych.

* * *

Obecna generacja mikroprocesorów 32-bitowych stanowi podstawę do budowania systemów komputerowych, porównywalnych pod względem szybkości działania i pojemności pamięci z niektórymi superminikomputerami, a nawet z dużymi komputerami. Ponieważ dla większości z nich nie ma jeszcze żadnych praktycznych doświadczeń, jest zbyt wcześnie, aby wyciągać wnioski dotyczące ich mocnych i słabych stron. Choć wstępne oceny dokonane przez firmę Intel mogą wskazywać na pewną wyższość mikroprocesora 80386, to jednak dotychczas posiadane dane nie są przekonujące i konieczne jest przeprowadzenie dokładniejszych badań.

Tłumaczył i opracował
JANUSZ ZALEWSKI

Tabela 1. Porównanie właściwości mikroprocesorów 80386 i MC68020

Właściwość	80386	MC68020
Częstotliwość zegarowa [MHz]	16	16,67
Liczba cykli zegara na rozkaz (średnio)	4,2	6,7
Szybkość operacji (megarozkazy na sekundę)	3,6	2,5
Liczba cykli zegarowych potrzebnych do translacji adresu wirtualnego na liniowy (najgorszy przypadek)	2	24
Czas obliczenia adresu [ns]	0—125	0—900
Mnożenie całkowitoliczbowe 32-bitowe [µs]	0,56—2,4	2,6
Dzielenie całkowitoliczbowe 32-bitowe [µs]	2,4	4,7
Szerokość pasma magistrali lokalnej [MB/s]	32	22

Tabela 2. Porównanie cykli odczytu i zapisu dla mikroprocesorów 89386, MC68020, WE32100 i NS32032 (MMU — jednostka zarządzenia pamięcią)

Procesor	Częstotliwość zegarowa [MHz]	Liczba cykli zapisu/odczytu	Czas [ns]
80386	16	4	250
MC 68020	16,67	6	360
MC 68020 + MMU	16,67	8	480
WE 32100	14	8	571
NS 32032	10	8	800
NS 32032 + MMU	10	10	1000

Tabela 3. Porównanie właściwości omawianych mikroprocesorów 32-bitowych

Właściwość	80386	MC68020	NS32332	WE32100	Z80000
Maksymalna częstotliwość zegarowa [MHz]	16	20	15	18	25
Liczba trybów adresowania	11	18	9	18	9
Liczba końcówek obudowy	132	120	84	132	64
Multipleksowane szyny adresów i danych	Nie	Nie	Tak	Nie	Tak
Liczba 32-bitowych rejestrów roboczych	8	16	8	8	16
Liczba stopni zakładkowania	6	3	3	4	6
Wbudowana jednostka zarządzania pamięcią	Tak	Nie	Nie	Nie	Tak
Wbudowana pamięć podręczna (liczba bajtów)	Nie	Tak (256)	Nie	Tak (256)	Tak (256)
Segmentowana pamięć	Tak	Nie	Nie	Tak	Tak
Stronicowanie pamięci (rozmiar strony w KB)	Tak (4)	Tak (do 32)	Tak (0,5)	Tak (2)	Tak (1)

LITERATURA

[1] Agraz-Guerena J. et al.: A Second Generation 32-Bit CMOS Microprocessor. COMPCON'84

[2] American Telegraph and Telephone: WE32100 Microprocessor Data Sheet, No. 105059026, October 1985

[3] Fuccio M. L., Ng B.: Hardware Architecture Considerations in the WE32100 Chip Set. IEEE MICRO, Vol. 6, No. 2, pp. 29—46, April 1986

[4] MacGregor D., Mothercole D., Moyer B.: The Motorola MC 68020, IEEE Micro, Vol. 4, No. 4, pp. 101—118, August 1984.

[5] Motorola Inc.: MC68020 32-Bit Microprocessor User's Manual. Prentice-Hall, Englewood Cliffs (NJ) 1984.

[6] National Semiconductor Corp.: NS 32332 32-Bit Advanced Microprocessor with Virtual Memory, TL/EE/8673, October 1985.

[7] Phillips D.: The Z80000 Microprocessor. IEEE Micro, Vol. 5, No. 6, pp. 23—36, December 1985.

[8] Thompson R., Uberoi A.: Processor Offers Code Compatibility VAX-Like Architecture. Computer Design, pp. 76—80, 1 January 1986.

Daniel Tabak: Reduced Instruction Set Computer RISC Architecture. Research Studies Press (Wiley), 1987

Od pojawienia się — w połowie lat czterdziestych — pierwszych komputerów o architekturze von Neumanna trwają nieustanne wysiłki, zarówno w świecie akademickim jak i w przemyśle, zmierzające do udoskonalenia architektury komputerów. W ciągu tego czasu wyłoniło się wiele różnych kierunków i związanych z nimi projektów, na przykład, wymieniając tylko najnowsze: architektury sterowane przepływem danych (ang. data flow architectures), rekonfigurowalne architektury dynamiczne i — przedmiot tej książki — architektura RISC (komputery o zredukowanej liście rozkazów). W książce opisano zasady architektury RISC i przedstawiono wszystkie jej eksperymentalne i przemysłowe realizacje, znane do czerwca 1986 roku.

Spis treści: Wprowadzenie (koncepcja architektury RISC, wady i zalety, rys historyczny); Berkeley RISC I i II (architektura, organizacja i realizacja); Ocena działania komputera RISC; Opis innych komputerów RISC (prototypy doświadczalne, Stanford MIPS, IBM 801, GMU MIRIS, systemy przemysłowe, Pyramid, Ridge, Acorn ARM, Inmos Transputer, IBM 6150 RT PC, HP 3000 Series 930 i 950, Metaforth MF 1600, Farchild Clipper, porównanie); Komputer jednorozkazowy.

(J.Z.)

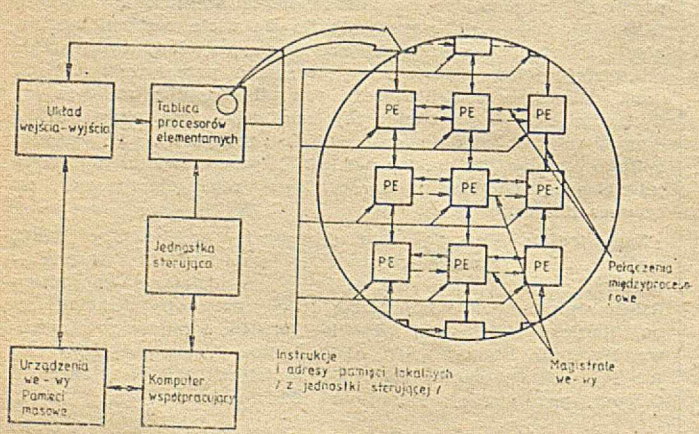
JACEK OWCZARCZYK
MACIEJ STOLARSKI
Instytut Podstaw Informatyki PAN
EWA WOŹNIAK
Instytut Maszyn Informatycznych
Warszawa

Procesory tablicowe — współbieżne systemy przetwarzania obrazów

Koncepcja systemu cyfrowego, złożonego ze stosunkowo dużej liczby prostych procesorów połączonych w regularną dwuwymiarową tablicę, powstała już prawie trzydzieści lat temu [17]. Dopiero jednak w ostatnich latach architektura taka stała się przedmiotem bardzo dużego zainteresowania i jest obecnie uważana za jeden z głównych kierunków zwiększenia mocy obliczeniowej w komputerach nowej generacji. Systemy komputerowe wykorzystujące tablicę procesorów nazywane są procesorami tablicowymi (ang. array processors).

Jednym z głównych zastosowań procesorów tablicowych jest cyfrowe przetwarzanie obrazów [9]. Przyczyną tej sytuacji jest doskonała zgodność strukturalna takiego systemu obliczeniowego z cyfrową postacią obrazów (są one reprezentowane dwuwymiarową regularną tablicą), a także zgodność topologii połączeń procesorów elementarnych (lokalne połączenia między sąsiednimi procesorami) ze strukturą obliczeń w operacjach lokalnych (najważniejsza klasa obliczeń w komputerowym przetwarzaniu obrazów). Ta wyjątkowa przydatność procesorów tablicowych do przetwarzania obrazów powoduje, że traktuje się je często jako specjalizowane procesory, a nie zawsze dostrzega się ich możliwości również w architekturach współbieżnych procesorów arytmetycznych dla komputerów uniwersalnych.

Atrakcyjność procesorów tablicowych polega również na możliwości ich praktycznej realizacji z elementów wykonanych w mniej nowoczesnych technologiach. Ich znaczne moce obliczeniowe mogą być uzyskane nie tylko dzięki korzystaniu z pojedynczych ultraszybkich układów arytmetycznych i pamięci, lecz także przez zwielokrotnienie liczby procesorów pracujących współbieżnie. Pamiętając więc o pewnej możliwej uniwersalności zastosowań procesorów tablicowych, w artykule omówiono je głównie pod kątem ich jak najlepszego dostosowania do organizacji obliczeń i przepływu danych wynikających z typowych algorytmów w cyfrowym przetwarzaniu obrazów.



Rys. 1. Schemat blokowy procesora tablicowego

ARCHITEKTURA PROCESORA TABLICOWEGO

Procesory tablicowe należą do systemów wieloprocesorowych typu SIMD (Single Instruction stream — Multiple Data stream; pojedynczy strumień instrukcji — wielokrotny strumień danych). Cechą charakterystyczną takiego systemu jest m. in. jedna wspólna dla wszystkich procesorów jednostka sterująca. Każdy z procesorów elementarnych jest wyposażony w pamięć lokalną i oddzielone rejestry. W systemach typu SIMD mogą być stosowane różne typy połączeń. W procesorach tablicowych z reguły każdy procesor jest bezpośrednio połączony z czterema lub ośmioma procesorami sąsiednimi.

Podstawowymi elementami procesora tablicowego (rys. 1) są: tablica procesorów elementarnych, układ wejścia-wyjścia, jednostka sterująca oraz sprzęg z współpracującym systemem komputerowym.

PROCESOR ELEMENTARNY

Cechą charakterystyczną procesorów elementarnych, stosowanych we współczesnych procesorach tablicowych, są jednobitowe układy arytmetyczno-logiczne z sekwencyjną realizacją obliczeń również dla złożonych jednostek informacji (bajt, słowo). Stosowanie arytmetyki jednobitowej jest podytkowane przede wszystkim koniecznością zredukowania liczby połączeń międzyprocesorowych. Taki sposób realizacji obliczeń, chociaż mniej efektywny w porównaniu z równoległym, ma jednak pewne zalety. Przede wszystkim jest nią duża elastyczność, pożądana zwłaszcza w komputerowym przetwarzaniu obrazów, gdzie liczba bitów dla podstawowej tu jednostki informacji — elementu obrazu — jest różna w różnych zastosowaniach. Inną zaletą jest możliwość częściowego przetwarzania słowa przez bezpośrednie testowanie znaku liczby, bez konieczności transmisji całego słowa [2].

Jednostka arytmetyczno-logiczna procesora elementarnego zawiera przeważnie sumator jednobitowy. Procesor taki jest wyposażony w kilka jednobitowych rejestrów, lokalną pamięć i układy przełączające umożliwiające selektywne przesyłanie (odbieranie) danych do (z) sąsiednich procesorów. Użycie w procesorze niezależnego od układów arytmetyczno-logicznych rejestru wejścia-wyjścia pozwala na realizację operacji przesyłania danych do (z) tablicy. współbieżnie z przetwarzaniem.

Architektura procesora elementarnego jest optymalizowana pod kątem struktury planowanych obliczeń, a w procesorach tablicowych, stosowanych przede wszystkim do przetwarzania obrazów, jest ona wynikiem kompromisu między optymalną organizacją do realizacji operacji lokalnych na obrazach binarnych a możliwością efektywnego wykonywania operacji arytmetycznych, przy czym celem nadrzędnym jest możliwie jak najprostsza struktura procesora.

UKŁAD WEJŚCIA-WYJŚCIA

Zadaniem układu we-wy jest buforowanie danych oraz wykonywanie zmian w ich formatach. W typowych syste-

mach przetwarzania obrazów wykorzystuje się zazwyczaj szeregową transmisję danych. W układzie we-wy elementy obrazów są „ładowane" szeregowo do rejestrów przesuwnych (szeregowo-równoległych), a następnie równolegle (dla całej kolumny lub wiersza) do tablicy procesorów elementarnych.

Przesyłanie danych do tablicy może odbywać się według różnych schematów:

- tylko skrajny wiersz (kolumna) otrzymują dane z zewnątrz; tablica jest zapełniana przez propagację danych (równoległe przesyłanie do kolejnych wierszy),
- dzięki dodatkowym magistralom istnieje możliwość równoległego wypełniania zadanego wiersza (kolumny),
- jednobitowa tablica danych jest równolegle (jednocześnie) przesyłana do wszystkich procesorów elementarnych.

W paktyce najczęściej stosuje się sposób pierwszy lub drugi; trzeci natomiast jest używany w sytuacjach, gdy dane są wcześniej przesyłane do rejestrów we-wy sposobem pierwszym, a następnie (w jednym cyklu) — z tych rejestrów do pamięci lokalnych procesorów elementarnych. Sposób trzeci może być używany także w sytuacjach, gdy procesor tablicowy współpracuje z dwuwymiarową tablicą sensorów.

W algorytmach przetwarzania obrazów znaczną część stanowią operacje we-wy, a więc transmisje danych do (z) tablicy powinna odbywać się współbieżnie z procesem przetwarzania.

Zagadnieniem ściśle związanym z operacjami we-wy jest ładowanie obrazów cyfrowych w tych wypadkach, gdy ich wymiary przekraczają wymiary tablicy procesorów, co jest zresztą dosyć typową sytuacją. Istnieje już obecnie procesor tablicowy z tablicą o wymiarach 128×128, ale i tak nie pokrywa on pełnych wymiarów obrazów cyfrowych w konkretnych zastosowaniach. Zagadnienie to jest często dyskutowane w kontekście szybkości dostępu do elementów obrazu z żądanego otoczenia dla efektywnej realizacji operacji lokalnych [3, 5, 14].

W sytuacji, gdy tablica procesorów ma wymiary $m \times m$, a obraz cyfrowy $n \times n$ ($n > m$), najczęściej stosuje się dwa sposoby ładowania obrazu do pamięci lokalnych procesorów elementarnych.

Pierwszy z nich polega na ładowaniu i przetwarzaniu segmentów obrazu o wymiarach $m \times m$ w kolejnych etapach (rys. 2a). Ponieważ jednak taka procedura uniemożliwia wykonywanie operacji lokalnych dla punktów, których otoczenie wykracza poza segment, dlatego też należy bądź przechowywać częściowe wyniki, bądź stosować metodę częściowego nakładania na siebie kolejnych segmentów.

Rys. 2. Metody umieszczania obrazu cyfrowego (rysunek o wymiarach 12×12) w pamięciach lokalnych 4×4 procesorów elementarnych (na rysunku pamięci lokalne procesorów oznaczone są literami A, ..., P)

Drugie, alternatywne rozwiązanie polega natomiast na ładowaniu do pamięci lokalnych bloku obrazu o wymiarach $n/m \times n/m$, w taki sposób, jak to zilustrowano na rysunku 2b [3]. Podczas wykonywania operacji lokalnych każda para sąsiednich procesorów musi wymienić informację jedynie o wartościach położonych wzdłuż wspólnej granicy. Sposób ten wymaga jednak stosowania pamięci lokalnej procesorów elementarnych o odpowiednio dużej pojemności.

JEDNOSTKA STERUJĄCA

Najważniejsze zadania, jakie spełnia w procesorze tablicowym jednostka sterująca, są następujące:

- dekodowanie mikroinstrukcji i generowanie odpowiednich sygnałów sterujących dla procesorów elementarnych,
- sterowanie topologią połączeń skrajnych (brzegowych) procesorów tablicy i ustalanie dla tych połączeń wymaganych poziomów sygnałów,
- sterowanie przepływem danych do (z) tablicy (kontrola operacji we-wy),
- wyznaczanie wartości pewnych stałych, niezbędnych do określania adresów globalnych pamięci lokalnych procesorów elementarnych, a także do wyznaczania pętli programowych.

Pożądaną cechą topologii połączeń dla skrajnych procesorów jak i organizacji układów arytmetyczno-logicznych w procesorach elementarnych jest m. in. możliwość formowania różnego rodzaju równoległych układów arytmetycznych w taki sposób, aby na przykład tablica jednobitowych procesorów o wymiarach 16×16 mogła być również używana jako szesnaście 16-bitowych sumatorów lub jako cztery sumatory 64-bitowe.

WYBRANE PROCESORY TABLICOWE

Spośród wielu projektów procesorów tablicowych zrealizowanych w ostatnich latach najbardziej znaczącymi są trzy następujące:

— CLIP 4 (Cellular-Logic Image Processor) [4],
— DAP (Distributed Array Processor) [12],
— MPP (Massively Parallel Processor) [1, 2].

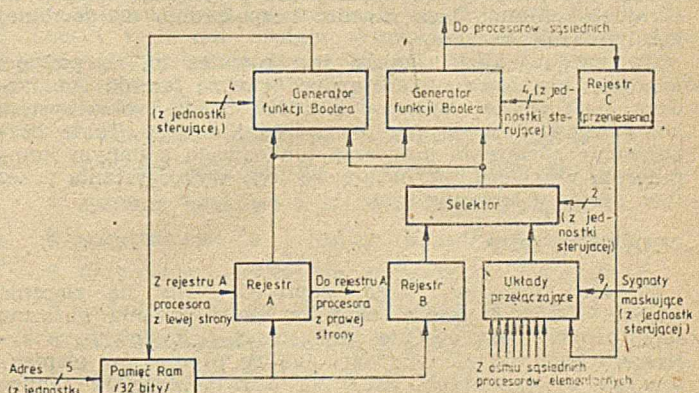
Tabela. Szybkości obliczeń wybranych procesorów (w milionach operacji na sekundę)

Operacja	CLIP4	DAP	MPP
Dodawanie stałoprzecinkowe (ośmiobitowe argumenty)	41	759	6554
Mnożenie stałoprzecinkowe (ośmiobitowe argumenty)	2,5	99,4	1862
Operacje lokalne dla obrazów wielotonalnych (256 poziomów) — splot, jądro 3x3	1	12	213
Operacje lokalne na obrazach	369	2276	8192

W tabeli podano szybkości obliczeń uzyskiwane w tych procesorach dla wybranych operacji (szybkości podane są w milionach operacji na sekundę).

PROCESOR CLIP 4

Jest to czwarty z serii procesorów tablicowych, jakie zostały zaprojektowane i wykonane na Uniwersytecie w Londynie. Architektura jego jest ukierunkowana przede wszystkim na efektywne wykonywanie operacji lokalnych na obrazach binarnych. Świadczy o tym zarówno topologia wzajemnych połączeń procesorów elementarnych (każdy z nich połączony jest z ośmioma sąsiednimi procesorami), jak i struktura układów logicznych, umożliwiająca w jednym cyklu operacje lokalne (3×3). W tej realizacji znacznie dłużej wykonywane są operacje arytmetyczne — np. mnożenie wymaga aż 196 instrukcji. Ponadto bardzo mała pojemność pamięci lokalnej (32 bity) stwarza istotne trudności przy przetwarzaniu obrazów wielotonalnych.



Rys. 3. Schemat blokowy procesora elementarnego systemu CLIP4

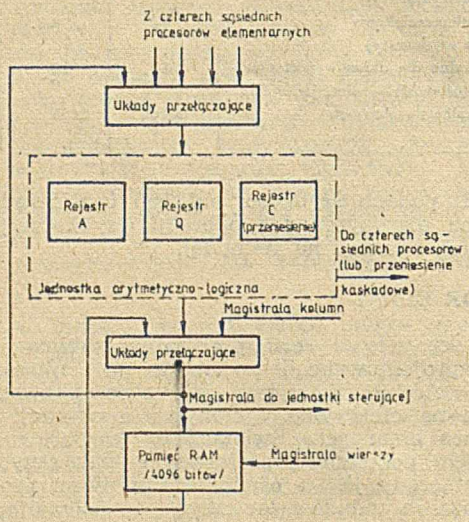
Tablica zawiera 96 × 96 procesorów elementarnych. Schemat blokowy pojedynczego procesora przedstawiono na rysunku 3. Jego głównymi składnikami są dwa generatory funkcji Boole'a, trzy jednobitowe rejestry A, B, C, układy przełączające oraz pamięć RAM. Rejestr A używany jest do przesyłania danych do (z) tablicy, a rejestr C jest rejestrem przeniesienia. Operacje we-wy nie są tu współbieżne z przetwarzaniem. Tablica procesorów jest zbudowana z układów scalonych wykonanych w technologii NMOS. Każdy układ zawiera osiem procesorów (łącznie z ich pamięciami lokalnymi).

Procesor tablicowy CLIP 4 dzięki odpowiednim układom umożliwia sprzętową realizację pewnych globalnych operacji związanych z przetwarzaniem obrazów. Układ we-wy umożliwia również bezpośrednie przetwarzanie wybranego fragmentu obrazu z kamery telewizyjnej.

PROCESOR DAP

Procesor tablicowy DAP został opracowany przez firmę ICL. Jako jedyny z trójki omawianych tu procesorów tablicowych nie został on zaprojektowany wyłącznie pod kątem przetwarzania obrazów, lecz raczej z myślą o znacznie większym kręgu zastosowań.

Tablica procesorów ma wymiary 64 × 64, a każdy z procesorów elementarnych jest połączony z czterema sąsiednimi. Schemat blokowy procesora elementarnego przedstawiono na rysunku 4. Głównymi elementami tego procesora są: jednobitowy sumator, trzy jednobitowe rejestry A, Q, C oraz pamięć RAM o pojemności 4096 bitów. Rejestr Q pełni funkcję akumulatora, rejestr C jest rejestrem przeniesienia, natomiast rejestr A pełnił m.in. funkcję rejestru maskującego (dla niektórych instrukcji zapis do pamięci RAM jest możliwy jedynie wówczas, gdy jego stan jest 1). Procesory elementarne są zbudowane z układów scalonych TTL.



Rys. 4. Schemat blokowy procesora elementarnego systemu DAP

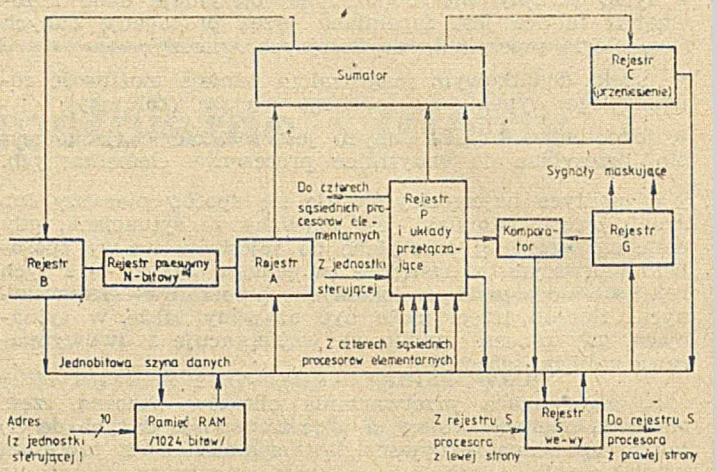
Ważnym elementem procesora DAP są dodatkowe magistrale (wierszy i kolumn) umożliwiające transmisję danych do (z) tablicy nie tylko przez propagację danych ze skrajnej kolumny, lecz również bezpośrednio do dowolnej kolumny lub wiersza.

Unikatowym rozwiązaniem jest również to, że pamięci lokalne procesorów elementarnych tworzą łącznie standardowy blok pamięci systemu ICL 2900, dzięki czemu system komputerowy ma do niej bezpośredni dostęp. Takie bezpośrednie sprzężenie procesora tablicowego z danym komputerem wyklucza już możliwość jego wykorzystania przez inne systemy komputerowe.

PROCESOR MPP

Procesor tablicowy MPP, zaprojektowany na zlecenie agencji NASA i przeznaczony do przetwarzania obrazów satelitarnych, jest obecnie najbardziej rozbudowanym systemem tego rodzaju. Zawiera tablicę procesorów elementarnych o wymiarach 128 × 128 oraz rezerwową tablicę 128 × 4, która może zastąpić dowolny blok tablicy głównej w wypadku awarii jednego z jej procesorów.

Schemat blokowy procesora elementarnego przedstawiono na rysunku 5. Zawiera on jednobitowy sumator, sześć rejestrów (A, B, C, G, P i S), rejestr przesuwny o programowanej długości (upraszczający realizację mnożenia i dzielenia), pamięć RAM o pojemności 1024 bitów (z możliwością rozszerzenia do 65536 bitów), wewnętrzną magistralę i dodatkowe układy logiczne i przełączające. Każdy z procesorów elementarnych jest połączony z czterema sąsiednimi.



Rys. 5. Schemat blokowy procesora elementarnego systemu MPP
*** N = 2, 6, 10, 14, 18, 22, 26, 30**

Sumator, rejestry A, B i C (rejestr przeniesienia) oraz rejestr przesuwny są używane podczas wykonywania operacji arytmetycznych, przy czym długość ich argumentów jest określona przez łączną długość rejestru przesuwnego i rejestrów A i B. Mnożenie jest wykonywane jako seria dodawań, a wyniki częściowe są przemieszczane w rejestrze przesuwnym.

Rejestr S jest używany w operacjach we-wy. Ładowanie danych do tablicy odbywa się równolegle dla skrajnej kolumny procesorów, a następnie zawartość rejestrów S jest cyklicznie przesyłana do kolumn kolejnych. Przesyłanie danych z rejestrów S do pamięci RAM jest wykonywane dla całej tablicy w jednym cyklu. Operacje przesyłania danych do (z) tablicy są współbieżne z procesem przetwarzania.

Rejestr G jest rejestrem maskującym — operacje, które mogą być maskowane, są wykonywane jedynie wówczas, gdy stan rejestru G jest równy 1.

Układ we-wy procesora MPP zawiera dwie pamięci buforowe o pojemności 2,5 MB, wykorzystywane podczas zmiany formatu danych. Istnieją także możliwości programowego ustawiania różnych topologii połączeń dla skrajnych procesorów tablicy. Należy zaznaczyć, że procesory elementarne mogą nawet utworzyć wektor składający się z 16 384 elementów.

Procesor MPP posiada także bardzo rozbudowaną jednostkę sterującą, a całym systemem steruje komputer PDP 11/45.

Tablica procesorów wykonana jest z układów VLSI. Każdy układ zawiera osiem procesorów elementarnych (4 × 2). Ich pamięci lokalne są zbudowane z typowych układów o organizacji 1024 × 4 bity (jeden układ stanowi pamięć lokalną dla czterech procesorów).

NOWE TENDENCJE W ARCHITEKTURZE PROCESORÓW TABLICOWYCH

Omówione procesory mają pionierski charakter; oparte na nich systemy umożliwiają zarówno praktyczną ocenę efektywności przyjętych rozwiązań, jak i formułowanie ogólnych zasad programowania dla tego rodzaju procesorów. Niemniej jednak nie wyczerpują one jeszcze wszystkich możliwych tendencji występujących w architekturze procesorów tablicowych.

W ostatnich latach pojawiło się wiele nowych projektów procesorów tablicowych, a niektóre z nich zasługują na szczególną uwagę. Na przykład procesor tablicowy BASE [13] jest bardzo dobrze dostosowany do realizacji rekurencyjnych operacji lokalnych na obrazach binarnych. Inny

procesor, LIPP (ang. Linköping Image Parallel Processor [5], ma dodatkowe rozbudowane magistrale umożliwiające optymalne przesyłanie danych do tablicy przy realizacji operacji lokalnych. Wykorzystuje on również inny niż stosowany dotychczas w procesorach tablicowych sposób reprezentacji liczb (znak-logarytm). Procesor AAP (ang. Adaptive Array Processor), opisany w [8], umożliwia stosowanie wielowarstwowych tablic dzięki używaniu połączeń nie tylko z ośmioma sąsiednimi procesorami, lecz również z procesorami — „górnym" i „dolnym".

Ta ostatnia właściwość procesora tablicowego, tzn. trójwymiarowa organizacja tablic, jest szczególnie użyteczna w takich zastosowaniach jak przetwarzanie obrazów oparte na hierarchicznych strukturach danych, np. w postrzeganiu komputerowym (ang. computer vision). Z myślą o tych zastosowaniach projektuje się wielowarstwowe, piramidalne procesory tablicowe [15].

Spotykana często w poprzednich latach tendencja planowania budowy w kolejnych wersjach procesora tablicowego o coraz większych wymiarach tablic wydaje się obecnie zanikać. Spowodowane jest to nie tylko samymi trudnościami technicznymi, związanymi m. in. z zapewnieniem synchronicznej pracy w bardzo dużych tablicach (nierównomierne czasy propagacji sygnałów zegarowych i kontrolnych), ponieważ można temu zaradzić [6], lecz przede wszystkim z ciągle jeszcze bardzo dużymi kosztami takich systemów. Tak więc brak motywacji ekonomicznej z jednej strony, a z drugiej przekonanie, iż dzięki postępowi w technologii moce obliczeniowe procesorów tablicowych o małych (16×16) lub średnich (32×32, 64×64) tablicach będą coraz większe, skłania do ogólnego wniosku, że dla większości zastosowań (w tym także dla przetwarzania obrazów) są to obecnie optymalne wymiary tablic [3, 5, 11]. W procesorach z takimi tablicami szczególnego znaczenia nabiera właściwa organizacja magistrali przesyłowych, umożliwiających efektywne przetwarzanie dużych tablic danych (obrazów cyfrowych).

Dzięki postępowi w technologii VLSI, a ostatnio UVLSI, budowa procesorów tablicowych staje się coraz mniej skomplikowana. Przykładem może być wspomniany wcześniej procesor AAP, do którego konstrukcji użyto układów scalonych zawierających 64 (8×8) procesory elementarne, o względnie rozbudowanych układach arytmetyczno-logicznych, łącznie z ich stosunkowo niewielką (96 bitów) pamięcią lokalną.

Dotychczas stosowane technologie zmierzają do upakowania w pojedynczym układzie scalonym możliwie jak największej liczby kompletnych (ewentualnie bez pamięci) procesorów elementarnych. Zupełnie odmienną technologię integracji procesorów elementarnych zaproponowano natomiast w pracy [7]; poszczególne elementy — akumulatory, komparatory, pamięci — są wykonywane na oddzielnych płytkach krzemu. Kompletną tablicę tworzy się przez ułożenie kolejnych struktur, jedna na drugiej. Połączenia między poszczególnymi warstwami są wykonywane specjalną techniką (ang. microbridge interconnection). Zaletą takiej trójwymiarowej organizacji jest duża modułowość; architektura procesora tablicowego może być dostosowywana do konkretnych zastosowań przez dobór odpowiedniego zestawu warstw.

* * *

Opisane procesory tablicowe stanowią obecnie jedną z najbardziej rozwijanych architektur w komputerach nowej generacji. Istnieje wiele przyczyn, które uzasadniają zainteresowanie właśnie taką architekturą; większość z nich omówiono w tym artykule. Dodatkowo można podać, że system komputerowy oparty na procesorze tablicowym wielokrotnie przewyższa zarówno system o klasycznej architekturze, jak i sieć wieloprocesorową typu MIMD pod względem stopnia wykorzystywania elementów czynnych systemu (bramki procesorów i rejestry pamięci biorące udział w obliczeniach) [16].

Bezpośrednim przejawem tego zainteresowania jest bardzo duża liczba projektów procesorów tablicowych. Projekt takiego procesora jest również opracowywany w Instytucie Podstaw Informatyki PAN [10].

LITERATURA

[1] Batcher K.: Design of a Massivelly Parallel Processor. IEEE Trans. on Computers, Vol. C-29, No. 9, pp. 836—840, 1980

[2] Batcher K.: Bit-serial parallel processing systems. IEEE Trans. on Computers, Vol. C-31, No. 5, pp. 377—384, 1982

[3] Danielsson P., Levialdi S.: Computer architectures for pictorial information systems. Computer, No. 11, pp. 53—67, 1981

[4] Duff M.: Parallel processors for digital image processing. Advances in Digital Image Processing, Stucki P. (ed.), pp. 265—276, Plenum Press, New York, 1979

[5] Ericsson T., Danielsson P.: LIPP — a SIMD multiprocessor architecture for image processing. Pp. 395—400, proc. 10th Annual Symposium on Computer Architecture, Stockholm 1983

[6] Fisher A., Kung H.: Synchronizing large VLSI processor arrays. IEEE Trans. on Computers, Vol. C-34, No. 8, pp. 734—740, 1985

[7] Grinberg J., Nudd G., Etchells R.: A cellular VLSI architecture. Computer, No. 1, pp. 69—81, 1984

[8] Kidode M.: Image processing machines in Japan. Computer, No. 1, pp. 68—80, 1983

[9] Owczarczyk J., Stolarski M., Woźniak E.: Architektura współbieżnych sytemów przetwarzania obrazów. »Informatyka«, nr 5, 1986

[10] Owczarczyk J., Stolarski M.: Projekt wstępny procesora tablicowego TRAP. Prace IPI PAN, Warszawa 1986 (przygotowywane do druku)

[11] Preston K.: Cellular logic computers for pattern recognition. Computer, No. 1, pp. 36—47, 1983

[12] Reddaway S.: The DAP approach. Infotech State of Art Report on Supercomputers, Vol. 2, pp. 836—840, 1979

[13] Reeves A.: A systematically designed binary array processor. IEEE Trans. on Computers, Vol. C-29, No. 4, pp. 278—287, 1980

[14] Rosenfeld A.: Parallel image processing using cellular arrays. Computer, No. 1, pp. 14—20, 1983

[15] Tanimoto S.: A pyramidal approach to parallel processing. pp. 372—378, proc. 10th Annual Symposium on Computer Architecture, Stockholm, 1983

[16] Uhr L.: Comparing serial computers, arrays and networks using measures of„active resources". IEEE Trans. on Computers, Vol. C-31, No. 10, pp. 1022—1025, 1982

[17] Unger S.: A computer oriented towards spatial problems. Proc. IRE, Vol. 46, pp. 1744—1750, 1958.

CeBIT'87

Największa europejska ekspozycja technologii informatycznej, jaką jest od co najmniej 15 lat hanowerski CeBIT, odbędzie się w br. w dniach od 4 do 11 marca. Przy tej okazji należy przypomnieć, że począwszy od ubiegłego roku CeBIT został wydzielony z głównej ekspozycji Targów Hanowerskich w samodzielną, specjalistyczną imprezę, odbywającą się w marcu (o miesiąc wcześniej niż targi przemysłowe). Żywiołowy rozwój informatyki na początku obecnej dekady w wyniku rewolucji mikrokomputerowej spowodował bowiem wielokrotne zwiększenie liczby wystawców, a w konsekwencji taki rozrost CeBIT, że powierzchniowo nie mógł się on już zmieścić w dotychczasowych ramach targów przemysłowych.

Ekspozycja CeBIT obejmuje 12 największych hal wystawowych o łącznej powierzchni ok. 200 000 m². Udział w imprezie zgłosiło ponad 2000 wystawców z 30 krajów. Ekspozycja będzie podzielona na 8 podstawowych grup tematycznych zlokalizowanych w odrębnych halach. W ramach tych hal nastąpi grupowanie wystawców zgodnie ze specjalistyczną tematyką ich ekspozycji, co przy tak gigantycznej koncentracji ofert stanowić będzie istotne ułatwienie dla zwiedzających.

Podstawowa tematyka i lokalizacja ekspozycji:

- systemy informacyjne i biurowe (hala nr 1),
- systemy bankowe i zabezpieczenia (hala nr 2),
- oprogramowanie i usługi konsultacyjne (hala nr 3),
- urządzenia peryferyjne komputerów (hala nr 4),
- technika biurowa i organizacyjna (hale nr 4 i 5),
- mikrokomputery (hale nr 5, 6, 7 i 14),
- telekomunikacja (hale nr 13, 16 i 17),
- systemy CIM — CAD/CAM i rejestracja danych (hala nr 18).

JANUSZ SKOLIMOWSKI
Instytut Informatyki
Politechnika Warszawska

Specjalizowany system mikrokomputerowy SIGMA-3

System mikrokomputerowy SIGMA-3 jest przewidziany do tworzenia rodziny mikrokomputerów specjalizowanych do różnych zastosowań profesjonalnych. Stanowi on alternatywną propozycję wobec obecnej praktyki, podyktowanej koniecznością, a polegającej na wykorzystywaniu do celów profesjonalnych mikrokomputerów domowych, sprowadzanych z zagranicy lub produkowanych w znikomych ilościach w kraju. W artykule omówiono konstrukcję i oprogramowanie mikrokomputera SIGMA-3, wskazując podstawowe różnice między nim a mikrokomputerem domowym, tzn. typowym uniwersalnym mikrokomputerem powszechnego użytku bez pamięci dyskowej, jak np. Meritum I lub ZX Spectrum.

KONFIGURACJA SPRZĘTOWA

Mikrokomputer SIGMA-3 nie zawiera pamięci dyskowej, która obecnie w kraju jest kosztowna i trudno dostępna. W zakresie konfiguracji sprzętowej SIGMA-3 wykazuje następujące różnice w stosunku do komputera domowego:

- Pamięć stała jest zwiększona do 32 KB i zawiera między innymi oprogramowanie specjalistyczne z danej dziedziny zastosowań.
- W obudowie jednostki centralnej mieści się monitor ekranowy (monochromatyczny) i wszystkie zespoły komputera z zasilaczem. Jedynie klawiatura jest ruchoma (dołączona kablem). Ponadto w obudowie znajdują się miejsca rezerwowe na pakiety specjalne. Unika się w ten sposób rozmaitych przystawek i zewnętrznych połączeń kablowych.
- Układ współpracy z magnetofonem kasetowym zapewnia zwiększoną niezawodność transmisji (m. in. przez podwójną kontrolę parzystości).
- Klawiatura ma zwiększoną liczbę klawiszy, m. in. oddzielne zestawy: klawiszy numerycznych, klawiszy do redagowania tekstu na ekranie i klawiszy operacyjnych.

Układ wyświetlania zapewnia rozdzielczość 30 wierszy po 64 znaki oraz grafikę mozaikową o rozdzielczości 128 × × 90 elementów. Ekran jest dzielony na kilka okienek, których rozmiary zmieniają się w zależności od potrzeb. Można, na przykład, wyświetlać i poprawiać program w jednym okienku, widząc równocześnie w drugim wyniki próbnego działania tego programu. Różnorodne komunikaty o stanie systemu są wyświetlane w oddzielnym okienku systemowym i nie mieszają się z innymi obrazami.

STRUKTURA OPROGRAMOWANIA

Oprogramowanie systemu SIGMA-3 składa się z podsystemu programowania SIGMA oraz opisanych niżej programów. Podsystem programowania SIGMA pełni rolę systemu operacyjnego, przy czym jego możliwości są istotnie powiększone w stosunku do możliwości komputera domowego. Izoluje on użytkownika od szczegółów technicznych, takich jak kody szesnastkowe, porty wejściowo-wyjściowe, adresy pamięci itd. Ponadto zawiera translator konwersacyjnego języka programowania SIGMA-L. Podsystem SIGMA zajmuje 16 KB pamięci stałej.

W pamięci operacyjnej mikrokomputera SIGMA-3 mogą się znajdować równocześnie cztery następujące programy napisane w języku SIGMA-L:

1) **program podstawowy**, odpowiadający programowi, który w mikrokomputerze domowym występuje jako jedyny i może być wprowadzony z klawiatury lub kasety, poprawiony, wyświetlony, wydrukowany, zeskładowany i wykonany;

2) **program biblioteczny**, zawierający zbiór procedur użytecznych w danym rodzaju pracy. Umożliwia to wykonywanie wielu programów mających wspólne procedury, bez powielania tych procedur w poszczególnych programach;

3) **program specjalistyczny**, zapisany w pamięci stałej i zawierający najważniejsze procedury wyspecjalizowane dla danej dziedziny zastosowań;

4) **program operacyjny**, również zapisany w pamięci stałej, pozwalający na ładowanie, składowanie i zarządzanie programami i zbiorami danych oraz wykonywanie wielu innych czynności organizacyjnych, w łatwy, konwersacyjny sposób przy użyciu opisanych dalej metod interakcji.

Metody interakcji i katalog

SIGMA-3, podobnie jak inne mikrokomputery, współpracuje z użytkownikiem w sposób interakcyjny. Metody interakcji są tu jednak bardziej rozwinięte. Należą do nich: wybór z menu wyświetlonego na ekranie oraz wypełnianie rubryk wyświetlonego formularza. Obie metody pozwalają wykorzystywać efektywnie system (bez konieczności programowania go) przez użytkownika-specjalistę z innej dziedziny niż informatyka, a nawet przez użytkownika nie przeszkolonego. W wypadku tworzenia własnego oprogramowania użytkownik może pominąć wspomniane metody interakcji i komunikować się bezpośrednio z podsystemem oprogramowania SIGMA. Widzi on wówczas system przez język programowania wysokiego poziomu SIGMA-L. Nie ma natomiast możliwości ani potrzeby używania kodu maszynowego lub języka asemblera.

Ważną rolę w systemie SIGMA-3 pełni katalog. Jest to zbiór nazwanych danych (zwykle strukturowych), zorganizowany w pamięci operacyjnej, ale poza programami. Programy mogą tworzyć, zmieniać i w inny sposób wykorzystywać dane zawarte w katalogu. Katalog można ładować i składować niezależnie od programów. Pozwala to przetwarzać zbiór danych przez ciąg programów, przekazywać dane z jednego programu do drugiego itd.

Translator języka SIGMA-L

Translator jedynego dostępnego języka programowania jest zrealizowany jako kompilator przyrostowy. Oznacza to, że program wynikowy ma efektywność zbliżoną do programu kompilowanego (w odróżnieniu od interpretowanego programu w Basicu w komputerze domowym), ale równocześnie programowanie jest konwersacyjne, tzn. jest możliwe natychmiastowe wykonanie instrukcji lub programu, bez faz kompilacji, konsolidacji, ładowania itp. Postać źródłowa programu nie jest przechowywana w pamięci, lecz każdorazowo odtwarzana na podstawie postaci wynikowej, np. podczas poprawiania programu na ekranie.

Język SIGMA-L jest używany do tworzenia oprogramowania wbudowanego w pamięć stałą i do pisania programów użytkowych. Użytkownik systemu specjalizowanego, opartego na mikrokomputerze SIGMA-3, ma możliwość uzyskania oprogramowania specjalistycznego w pamięci stałej, dopasowanego do swoich indywidualnych potrzeb.

JĘZYK PROGRAMOWANIA SIGMA-L

Język SIGMA-L pod względem składni jest w zasadzie oparty na języku Basic. Ze względu na złożoną specjalizację systemu, dokładne trzymanie się standardu nie było konieczne. Wprowadzono zatem kilka zmian w składni, mających na celu jej uproszczenie i ujednolicenie, bez zmniejszenia mocy języka. Wprowadzono też wiele rozszerzeń związanych ze specyfiką systemu, np. dotyczących komunikacji przez okienka, użycia metod interakcji (menu i formularzy) itp. Najważniejszą innowacją jest wprowadzenie typu agregatowego danych na miejsce tradycyjnie stosowanych tablic.

Najważniejsze cechy struktur agregatowych są następujące:

- elementy agregatu mogą być różnych typów (można np. umieszczać liczby obok tekstów),
- elementem agregatu może być agregat niższego rzędu, co pozwala tworzyć struktury hierarchiczne o dowolnie wielu poziomach,
- liczba elementów aregatu zmienia się dynamicznie, agregat zajmuje w każdej chwili tyle pamięci, ile aktualnie potrzebuje,
- można wstawiać i usuwać elementy agregatu z początku, z końca i ze środka struktury,
- dostęp do elementów agregatu odbywa się przez indeksy, jak w tablicach, albo w przyspieszony sposób przez przesuwanie tzw. wskaźnika elementu aktualnego.

Przy użyciu agregatów można odwzorowywać rozmaite struktury danych występujących w innych językach programowania, jak tablice, stosy, kolejki, listy, rekordy, drzewa, a także struktury innego rodzaju, jak np. tablice dwuwymiarowe trójkątne, ważne m. in. w geodezji przy rozwiązywaniu symetrycznych układów równań.

* * *

System został opracowany w Instytucie Informatyki Politechniki Warszawskiej i obecnie w Zakładzie Doświadczalnym tego Instytutu produkowana jest prototypowa seria w wersji specjalizowanej dla potrzeb geodezji, nosząca nazwę GEO-3. Równocześnie w opracowaniu są wersje dla innych zastosowań, m. in. w medycynie i rolnictwie.

W profesjonalnych zastosowaniach mikrokomputerów występują zwykle inne wymagnia odnośnie sprzętu, jakości oprogramowania, specjalizacji oprogramowania i efektywności niż w zastosowaniach domowych lub szkolnych. Dlatego, obok niewątpliwej potrzeby rozwijania produkcji mikrokomputerów powszechnego użytku, występuje również potrzeba tworzenia efektywnych systemów mikrokomputerowych specjalizowanych dla różnych zastosowań profesjonalnych. Projekt SIGMA-3 jest krokiem w tym właśnie kierunku.

MAREK GONDZIO
Instytut Informatyki
Politechnika Warszawska

Sieci Petriego (2)

Przekształcenia i rodzaje sieci

W pierwszej części artykułu przedstawiono definicje sieci Petriego oraz podstawowych pojęć stosowanych w opisie i analizie sieci. Podano praktyczną interpretację terminów i przykłady ilustrujące zastosowanie tych sieci, jako narzędzia modelowania różnego rodzaju obiektów. Sformułowano również niektóre własności możliwe do badania za pomocą sieci Petriego oraz pokazano jak je badać, analizując graf znakowań osiągalnych.

Druga część artykułu zawiera wybrane modyfikacje i typowe zastosowania sieci Petriego. Precyzyjne omówienie różnych klas sieci Petriego wymagałoby znacznego rozbudowania używanej dotąd terminologii, przekraczającego ramy artykułu. Stąd większość zagadnień z tego tematu traktowana jest jedynie informacyjnie.

PRZEKSZTAŁCENIA SIECI PETRIEGO

Tworzenie sieciowego modelu określonego obiektu polega na ogół na automatycznym (lub chociaż półautomatycznym) wielokrotnym stosowaniu pewnych reguł, wyrażających z jednej strony sposób modelowania obiektu, a z drugiej — interpretację poszczególnych elementów modelu. Przykłady takich reguł pokazano w pierwszej części artykułu. Model zbudowany na podstawie takich reguł może być dość skomplikowany. Na analizę większych sieci trzeba na ogół poświęcić większe środki (wymagają one znacznego obszaru pamięci na reprezentację samej sieci i jej grafu znakowań osiagalnych). Z tego względu często poszukuje się takich możliwości upraszczania sieci, które:

- można wykonać automatycznie,
- nie zmieniają globalnej interpretacji całego modelu,
- zachowują bez zmiany pewne istotne własności danej sieci.

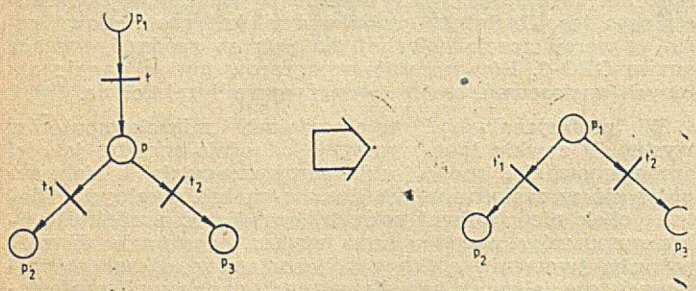
Konieczność uproszczeń stała się jedną z podstawowych przyczyn szerokiego rozwoju prac teoretycznych i badawczych, dotyczących transformacji i równoważności różnych sieci Petriego.

Typowymi przykładami przekształceń sieci Petriego są **redukcje**, które polegają na zmniejszeniu liczby miejsc, przejść lub krawędzi w danej sieci bez naruszania jej podstawowych własności. Poniżej scharakteryzowano trzy intuicyjnie proste redukcje. Są one przedstawione nieformalnie, ponieważ ich precyzyjne definicje są bardzo złożone. Zapis formalny obejmuje bowiem dokładny opis warunków, jakie muszą być spełnione, aby można było zastosować określoną redukcję, a także szczegółowy opis sposobu przekształcania sieci oraz pewne konsekwencje wykonania takiej operacji. Dokładne definicje podstawowych redukcji można znaleźć w [5].

- **Redukcję przez usunięcie miejsca** można wykonać tylko dla miejsc usuwalnych spełniających pewne, dokładnie zdefiniowane warunki [5]. W ogólnym wypadku wiąże się z nią złożona modyfikacja wszystkich elementów wymienionych w definicji sieci MPN, i to nie tylko zbioru miejsc, który zostaje zmniejszony o usuwane miejsce, ale także zbioru przejść, którego liczność może ulec zmniejszeniu, zwiększeniu lub pozostać bez zmiany, zbioru krawędzi i znakowania. Poza tym wykonanie tego przekształcenia sieci oznacza zmianę interpretacji niektórych przejść. W postaci zdefiniowanej w [5] usunięcie miejsca zachowuje m. in. bez zmiany takie własności sieci, jak ograniczoność, żywość i trwałość.

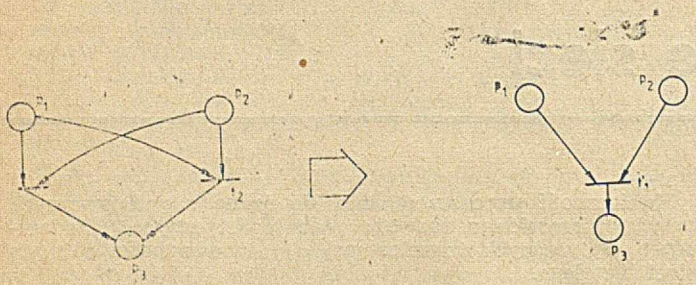
Przykład zastosowania tej redukcji przedstawia rys. 1. Z usunięciem miejsca **p** wiąże się także usunięcie przejścia t. Interpretacja (np. jakaś czynność) związana z usu-

niętym przejściem t zostaje przeniesiona do przejść wyjściowych usuwanego miejsca (t_1 it_2). Po przekształceniu każde z tych przejść ma już nową interpretację (t_1' t_2'), reprezentuje bowiem sekwencję czynności odpowiednio $t;t_1$ i $t;t_2$. Znaczenie miejsc p_1, p_2 i p_3 pozostaje oczywiście bez zmiany. Redukcja ta znajduje praktyczne zastosowanie przy „skracaniu" sekwencji miejsc i przejść połączonych pojedynczymi krawędziami. Kilkakrotne zastosowanie tej redukcji pozwala sprowadzić całą sekwencję do jednego przejścia, które przedstawia wtedy cały ciąg czynności reprezentowanych poprzednio przez usuwane przejścia. Jak to wynika z rys. 1, można nawet usunąć miejsce wyznaczające „koniec" takiej sekwencji.



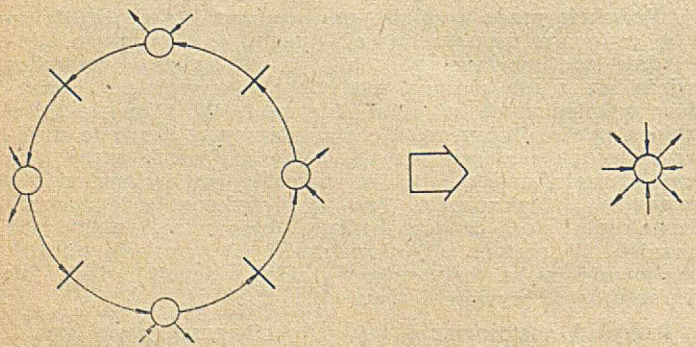
Rys. 1. Usunięcie miejsca

• **Redukcja przez kompresję przejść identycznych.** Przejściami identycznymi nazywa się przejścia, które mają takie same zbiory miejsc wejściowych i takie same zbiory miejsc wyjściowych. Bez naruszania podstawowych własności sieci wszystkie przejścia identyczne mogą być reprezentowane przez jedno (dowolne) z nich. W wypadku, gdy w danym modelu interpretacja usuniętych przejść jest istotna, można przyjąć, że wzbudzenie pozostawionego przejścia reprezentuje zajście jednego (nie wiadomo którego) zdarzenia spośród zdarzeń reprezentowanych przed przekształceniem przez wszystkie przejścia identyczne. Przykład zastosowania tej redukcji przedstawia rys. 2.



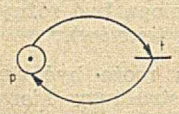
Rys. 2. Kompresja przejść identycznych

• **Redukcja przez usunięcie cyklu** dotyczy zamkniętej cyklicznej sekwencji miejsc i przejść, w której przejścia nie mają bezpośrednich połączeń z resztą sieci. Przedstawiona redukcja (rys. 3), w której cały cykl zostaje zastąpiony pojedynczym miejscem, zachowuje wprawdzie bez zmiany przedstawione formalne własności sieci (np. trwałość, ograniczoność, żywość [5]), ale znacznie ogranicza praktyczną interpretację modelu.



Rys. 3. Usunięcie cyklu

Głównym ograniczeniem stosowania tych reguł jest zmniejszanie przez redukcje reprezentatywności modelu. Uwzględnianie interpretacji przypisywanej modelowi jest praktycznie niemożliwe w automatycznych systemach redukcji sieci, natomiast nieskrępowane wykorzystanie redukcji może czasem prowadzić do pełnej utraty początkowego znaczenia modelu. W pracy [5] zastosowano iteracyjnie wszystkie możliwe redukcje w modelu systemu producentów-konsumentów synchronizujących swą współpracę za pomocą semafora. Początkowy model zawierał 18 miejsc i 15 przejść. Po przekształceniach model ten został zredukowany do najprostszej możliwej sieci (rys. 4). Można to żartobliwie skomentować, że zastosowanie redukcji przekształciło model systemu producentów-konsumentów w model reakcji chemicznej z katalizatorem (patrz pierwsza część artykułu).



Rys. 4. Najprostsza sieć Petriego

Ułatwienie analizy stanowi cel także innego przekształcenia sieci — dekompozycji. W uproszczeniu — **dekompozycja** polega na podziale sieci na podsieci [2], które mogą być niezależnie analizowane, a wyniki takich badań pozwalają wnioskować o pewnych własnościach całego modelu — np. o niezależności pewnych elementów składowych modelowanego obiektu.

KLASY SIECI PETRIEGO

Definiowanie nowych rodzajów sieci Petriego nastąpowało z reguły wtedy, gdy istniejące już rodzaje sieci okazały się niewystarczające lub niewygodne dla nowych zastosowań. Wprowadzane modyfikacje dotyczyły różnych elementów sieci, lecz zawsze zachowywały rozróżnienie między miejscami i przejściami oraz ideę ruchu znaczników w sieci. Nie wszystkie z proponowanych zmian znalazły szersze zastosowanie, jednak wiele na trwałe weszło do teorii i praktyki bardzo szeroko rozumianych obecnie sieci Petriego.

W charakterystyce każdej klasy sieci Petriego umownie wyróżniamy następujące dwa czynniki:

• **siłę modelowania** — rozumianą jako rozmiar (różnorodność) klasy obiektów, które można modelować za pomocą sieci danego rodzaju,

• **moc analityczną** — wyrażającą się liczbą i rodzajami własności, które w sieciach danej klasy można badań algorytmicznie.

Jak słusznie zauważył Keller [12], istnieje między nimi wzajemna zależność, która przy modyfikowaniu sieci wyraża się tym, że zwiększanie siły modelowania obniża moc analityczną i na ogół odwrotnie. Z tego względu w definicjach nowych klas sieci Petriego występują zarówno rozszerzenia, jak i zawężenia w stosunku do definicji sieci przedstawionej w pierwszej części artykułu. Przykładowe zawężenia dotyczą ograniczania zbioru krawędzi i ruchu znaczników w sieci. Zmniejszenie liczby krawędzi osiąga się na przykład poprzez eliminację przejść potencjalnie konfliktowych lub miejsc, które są jednocześnie wejściowe i wyjściowe dla określonego przejścia. Ruch znaczników może być z kolei ograniczony przez włączenie do definicji przejścia przygotowanego dodatkowego warunku, że żadne z miejsc wyjściowych tego przejścia nie może zawierać znacznika lub też przez związanie z każdym miejscem pojemności określającej maksymalną liczbę znaczników, jakie mogą znajdować się w danym miejscu. „Węższymi" w porównaniu do zwykłej sieci Petriego (zdefiniowanej w pierwszej części artykułu) są następujące modele:

• **maszyny stanów** (ang. state machines) — sieci, w których każde przejście ma dokładnie jedno miejsce wejściowe i dokładnie jedno miejsce wyjściowe,

• **grafy znakowane** (ang. marked graphs) — sieci, w których każde miejsce ma dokładnie jedno przejście wejściowe i dokładnie jedno przejście wyjściowe. Grafy znakowane nazywa się również grafami synchronizacji (ang. synchronization graphs) [8],

• **grafy wolnego wyboru** (ang. free-choice graphs) — sieci, w których miejsce mające więcej niż jedno przejście

wyjściowe, jest dla każdego z takich przejść jedynym miejscem wejściowym, czyli:

$$\forall p \in P : card(p^{\bullet}) > 1 \Rightarrow \forall t \in p^{\bullet} : {}^{\bullet}t = \{p\}$$

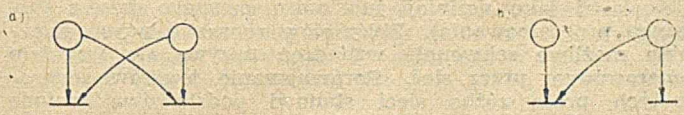
Przykład fragmentu sieci, która nie jest grafem wolnego wyboru, przedstawia rys. 5b. Zarówno maszyny stanu, jak i grafy znakowane stanowią podzbiory grafów wolnego wyboru,

Rys. 5. Ilustracja definicji grafu wolnego wyboru: a) zgodnie z definicją, b) niezgodnie z definicją

• **rozszerzone grafy wolnego wyboru** (ang. extended free-choice graphs) — sieci, w których zbiory przejść wyjściowych każdej pary miejsc, mających co najmniej jedno przejście wyjściowe wspólne — są identyczne, czyli:

$$\forall p_1, p_2 \in P : p_1^{\bullet} \cap p_2^{\bullet} \neq \Phi \Rightarrow p_1^{\bullet} = p_2^{\bullet}$$

Ilustracją tej definicji jest rys. 6.



Rys. 6. Ilustracja definicji rozszerzonego grafu wolnego wyboru: a) zgodnie z definicją, b) niezgodnie z definicją

Każda ze zdefiniowanych powyżej klas stanowi zawężenie w porównaniu do zwykłych sieci Petriego, ponieważ każda sieć dowolnej z tych klas jest siecią zwykłą. Łatwo natomiast znaleźć przykład zwykłej sieci Petriego, która nie spełnia postawionych powyżej wymagań.

Podstawowym rozszerzeniem jest wprowadzenie krawędzi wielokrotnych. Rozszerzenie to polega na przypisaniu każdej krawędzi wagi określającej liczbę znaczników „przenoszonych" przez daną krawędź podczas wzbudzania przejść. Tak zmodyfikowane sieci nazywamy dalej uogólnionymi sieciami Petriego [12]. Warto przytoczyć ich pełną definicję, ponieważ w wielu pracach [5, 9] właśnie ten rodzaj sieci uznaje się za podstawową klasę sieci Petriego.

Uogólnioną siecią Petriego (ang. Generalized Petri Net, GPN) nazywa się czwórkę $GPN = <P, T, A, W>$, gdzie **P**, **T** i **A** mają takie samo znaczenie jak w definicji zwykłej sieci Petriego (**PN**), a **W** jest funkcją określającą wagi krawędzi sieci:

$$W : A \rightarrow \{1, 2, ...\}$$

Przejście t sieci GPN jest przygotowane przez znakowanie M wtedy i tylko wtedy, gdy

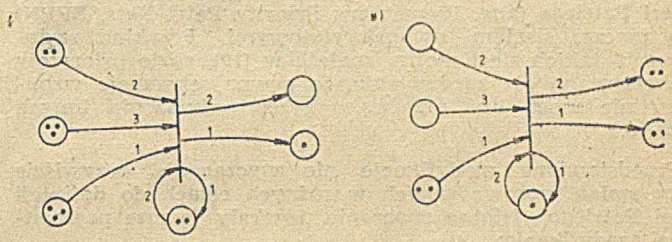
$$\forall p \in {}^{\bullet}t : M(p) \geq W(\langle p, t \rangle)$$

Wzbudzenie przejścia polega na pobraniu z każdego miejsca wejściowego liczby znaczników odpowiadającej wadze krawędzi łączącej dane miejsce z przejściem oraz na wprowadzeniu do każdego miejsca wyściowego liczby znaczników określonej wagą odpowiedniej krawędzi wyjściowej:

$$AP \in P : M \xrightarrow{t} M' \Rightarrow M^1(p) = \begin{cases} M(p) + W(\langle t, p \rangle), \text{ jeżeli } p \in t^{\bullet} - {}^{\bullet}t \\ M(p) - W(\langle p, t \rangle), \text{ jeżeli } p \in {}^{\bullet}t - t^{\bullet} \\ M(p) - W(\langle p, t \rangle) + W(\langle t, p \rangle), \\ \qquad \text{jeżeli } p \in {}^{\bullet}t \cap t^{\bullet} \\ M(p), \text{ jeżeli } p \notin {}^{\bullet}t \cup t^{\bullet} \end{cases}$$

Ilustrację tej formalnej definicji przedstawia rys. 7.

Sieci uogólnione umożliwiają czytelne modelowanie niektórych obiektów, trudnych do przedstawienia za pomocą zwykłych sieci Petriego. Umożliwiają one na przykład modelowanie za pomocą jednej sieci kilku identycznych elementów. W sieciach zwykłych dla każdego konsumenta (z systemu producentów-konsumentów) trzeba utworzyć osobny model, chociaż wszyscy konsumenci postępują według tego samego algorytmu. W sieciach uogólnionych konsumpcję można potraktować zbiorczo używając do tego celu krawędzi o wadze równej liczbie konsumentów. Model systemu zawiera wówczas tylko jedną sieć reprezentującą „zbiorowego konsumenta". Jest jednak wiele problemów, których modelowanie za pomocą **GPN** nie jest możliwe. Keller na przykład wykazał, że sieci uogólnione nie wystarczają do opracowania modelu sprawdzania warunku, czy wartość pewnej zmiennej **X** jest równa określonej stałej **k** ($X = k$?), jeżeli zmienna **X** może przyjmować dowolne wartości naturalne [12]. Inny przykład problemu, dla którego opracowanie modelu w klasie **GPN** nie jest możliwe, wskazał Kosaraju. Problem ten, występujący w literaturze [1] pod nazwą problemu Kosaraju, dotyczy modelowania systemu złożnego z dwóch producentów P_1 i P_2, dwóch konsumentów K_1 i K_2 oraz dwóch buforów B_1 i B_2. Obaj producenci (P_1 i P_2) umieszczają po jednym elemencie w buforach, odpowiednio B_1 i B_2. Pobieranie elementów z buforów B_1 i B_2 realizują odpowiednio konsumenci K_1 i K_2. Zarówno producenci, jak i konsumenci w sposób ciągły powtarzają wykonywane czynności. O ile producenci pracują niezależnie, to w przypadku konsumentów ma obowiązywać zasada, że K_2 może pracować tylko wtedy, gdy K_1 nie pracuje, czyli gdy bufor B_1 jest pusty.



Rys. 7. Wzbudzenie przejścia w sieci uogólnionej a) przed wzbudzeniem, b) po wzbudzeniu

Wygodnym narzędziem do opracowania modelu tak sformułowanego problemu są sieci, w których oprócz normalnych krawędzi występują także krawędzie wzbraniające. Ogólnie **siecią z krawędziami wzbraniającymi** (ang. Petri Net with Inhibitor arcs, PNI) nazywa się czwórkę $PNI = <P, T, A, I>$, w której **P**, **T**, **A** mają takie samo znaczenie jak w zwykłych sieciach Petriego, a **I** jest zbiorem krawędzi wzbraniających:

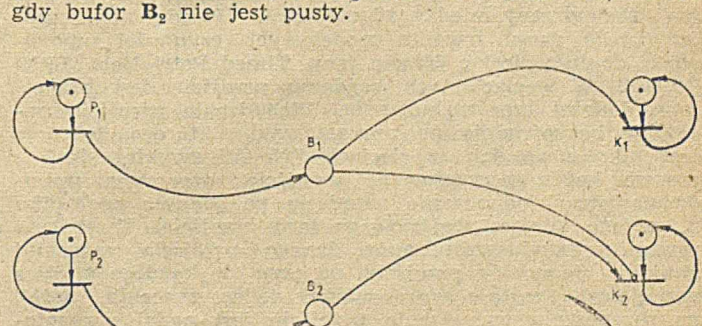
$I \subseteq P \times T$, przy czym **I**

W sieci **PNI** przejście t jest przygotowane przez dane znakowane M wtedy i tylko wtedy, gdy

$$\forall p \in {}^{\bullet}t : (\langle p, t \rangle \in A \Rightarrow M(p) \geq 1 \wedge \langle p, t \rangle \in I \Rightarrow M(p) = 0)$$

Krawędzie wzbraniające łączą więc z danym przejściem te miejsca, w których warunkiem przygotowania tego przejścia jest brak znacznika. Graficznie — krawędzie wzbraniające są wyróżnione małym okręgiem umieszczonym przy przejściu, tam gdzie w krawędziach zwykłych znajduje się strzałka. Definicja wzbudzenia przejścia jest dla sieci **PNI** taka sama, jak dla zwykłych sieci Petriego, ponieważ krawędzie wzbraniające z definicji nie mogą służyć do „przenoszenia" znaczników.

Przykładem sieci z krawędziami wzbraniającymi jest model problemu Kosaraju (rys. 8). Jedyna występująca w tym modelu krawędź wzbraniająca zapewnia, że jeżeli K_1 ma co „konsumować", to K_2 nie może pracować, nawet gdy bufor B_2 nie jest pusty.



Rys. 8. Problem Kosaraju

Sieci **PNI** stanowią oczywiście nadzbiór w stosunku do zwykłych sieci Petriego, jak jednak wykazano w [1], dla bezpiecznych sieci **PNI** można skonstruować równoważne im sieci zwykłe.

Następne rozszerzenie polega na wprowadzeniu do sieci priorytetów przejść [13].

Priorytetową siecią Petriego (ang. Priority Petri Net, PPN) nazywa się czwórkę $PPN = <P, T, A, O>$, gdzie. **P, T, A** mają. takie samo znaczenie jak w sieciach **PN**, natomiast **O** jest funkcją częściowo porządkującą zbiór **T** w ten sposób, że jeżeli $(t_i, t_j) \in O$, to t_i ma wyższy priorytet niż t_j.

W sieciach **PPN** inaczej musi być zdefiniowane przygotowanie albo wzbudzenie przejścia. Przyjmując, że definicja przygotowania przejścia pozostaje bez zmiany, do reguł określających wzbudzenie przejścia wprowadza się dodatkowy warunek. Przejście przygotowane przez znakowanie **M** może być wzbudzone, jeżeli nie ma innego przygotowanego przejścia o wyższym priorytecie. W klasie sieci priorytetowych wyróżnia się grupę sieci, dla której funkcja **O** dzieli zbiór przejść na takie dwa podzbiory T_X i T_Y, że:

$$T_X \cup T_Y = T \wedge (t_i, t_j) \in T_Y x T_Y \Leftrightarrow ((t_i, t_j) \in O \wedge t_i \neq t_j)$$

Sieci takie nazywa się **elementarnymi priorytetowymi sieciami Petriego** (ang. Elementary Priority Petri Nets, EPPN) lub inaczej sieciami dwupriorytetowymi. Przykład zastosowania takich sieci można znaleźć w [10], gdzie priorytety wykorzystano dla zapewnienia, że pewne sekwencje wzbudzeń nie mogą być przerwane przez wzbudzenia innych przejść.

Przedstawione modyfikacje nie wyczerpują oczywiście listy zmian wprowadzanych w różnych celach do definicji sieci Petriego. Istotne znaczenie uzyskały jeszcze następujące modyfikacje:

1. Przypisanie jakościowej informacji pojedynczym znacznikom, czyli wprowadzenie różnych typów (kolorów) znaczników. Uwzględnienie koloru znaczników pociąga za sobą zmianę definicji samego znakowania, które staje się funkcją dwuargumentową oraz przejścia przygotowanego i reguł wzbudzenia przejścia. W tej klasie sieci, nazwanej **sieciami kolorowanymi** (ang. Coloured Petri Nets, CPN), krawędziom nadaje się także nowe znaczenia przez przypisanie im kolorów znaczników, które mogą być po nich przenoszone [8].

2. Związanie z przejściami dodatkowego warunku wzbudzenia — podawanego w postaci wyrażenia logicznego. Wyrażenie to, zbudowane z określonych operatorów i predykatów, ma różne definicje. Ta klasa sieci nosi nazwę **sieci z predykatami** (ang. Petri Nets with Predicats, PNP) [8].

Każda z wprowadzonych modyfikacji była dotąd definiowana osobno i zawsze w odniesieniu do zwykłych sieci Petriego. W rzeczywistości bardzo często korzysta się z sieci, w których jednocześnie występują elementy charakterystyczne dla różnych klas. Dla uogólnionych, kolorowanych sieci z predykatami istnieje nawet odrębna nazwa, **PrT-sieci** (ang. Predicate Transition nets), powszechnie używana w wielu przeglądowych pracach na temat sieci Petriego (np. [8]).

W każdej z omawianych dotąd klas sieci Petriego wzbudzenie przejścia było operacją natychmiastową. Sieci te umożliwiają częściowe modelowanie uporządkowania zdarzeń w czasie, ale tylko na zasadzie relacji typu „przed/po" zachodzącej między zdarzeniami. Uwzględnienie rzeczywistego czasu trwania modelowych czynności umożliwiają dopiero **sieci z czasem** (ang. Timed Petri Nets, TPN) [17, 19]. W sieciach tych każdemu przejściu jest przypisany dodatni czas wzbudzania. Wzbudzenie przejścia następuje bezpośrednio po przygotowaniu, którego definicja jest taka sama jak w sieciach PN. W zwykłej sieci z czasem każde przygotowane przejście rozpoczyna wzbudzenie przez jednoczesne usunięcie po jednym znaczniku z każdego miejsca wejściowego tego przejścia. Przez czas równy czasowi wzbudzania danego przejścia znaczniki znajdują się w tym przejściu, po czym do każdego miejsca wyjściowego zostaje wprowadzony jeden znacznik. Jeżeli po rozpoczęciu wzbudzenia przejście jest nadal przygotowane, to natychmiast rozpoczyna się kolejna operacja wzbudzenia. Tak więc w sieciach z czasem nie tylko kilka różnych przejść może być wzbudzonych jednocześnie (tak jak dopuszcza to definicja sieci Petriego podana w [9]), ale nawet to samo przejście może się znaleźć w stanie wielokrotnego wzbudzenia. Przyjmuje się, że działanie sieci rozpoczyna się w chwili, gdy czas = 0. Dla tej chwili czasowej jest zdefiniowane znakowanie początkowe sieci. Jego definicja jest taka sama, jak w zwykłej sieci Petriego. Natomiast dla każdej niezerowej chwili czasowej opis sieci musi uwzględniać fakt, że znaczniki znajdują się nie tylko w miejscach, ale również w przejściach.

Analiza własności sieci z czasem musi więc być prowadzona całkiem innymi metodami (np. za pomocą tzw. opisówch chwilowych sieci [19]) niż analiza sieci zwykłych. Jest to oczywiście o wiele trudniejsze, ale z kolei sieci z czasem pozwalają uzyskać dokładniejsze dane o własnościach modelowanego obiektu.

Dla informatyki ważnym rodzajem sieci Petriego są **sieci etykietowane** (ang. Labeled Petri Nets, LPN). W sieciach tych z każdym przejściem związana jest etykieta stanowiąca symbol w pewnym alfabecie. Sieci etykietowane nie są odrębną klasą w dotychczasowym rozumieniu tego słowa, ponieważ w odróżnieniu od poprzednich modyfikacji przypisanie przejściom etykiet w postaci symboli jest jedynie pociągnięciem formalnym, nie mającym żadnego wpływu na ruch znaczników sieci. W sieciach LPN sekwencje wzbudzeń mogą być reprezentowane przez sekwencje etykiet (symboli) przypisanych wzbudzanym kolejno przejściom. Na tej zasadzie można użyć sieci etykietowanej jako definicji lub opisu pewnego języka (np. języka programowania). Zbiór słów repreznтujących wszystkie możliwe sekwencje wzbudzeń nazywa się **językiem generowanym przez sieć**. Porównywanie języków generowanych przez różne sieci stanowi podstawową metodę sprawdzania równoważności sieci.

ZASTOSOWANIA

Sieci Petriego wykorzystuje się przede wszystkim do modelowania różnego typu obiektów. Modeli sieciowych używa się jako formy opisu lub specyfikacji oraz jako narzędzia weryfikacji lub dowodzenia własności modelowanych obiektów. Sieci Petriego znalazły zastosowanie w takich dziedzinach, jak: logika, chemia, telekomunikacja, a nawet ekonomia i medycyna.

Większość praktycznych zastosowań występuje jednak w informatyce. Sieci Petriego używa się tu do modelowania:

- procesów obliczeniowych [8, 12]
- technik alokacji zasobów [1]
- mechanizmów synchronizacji procesów współbieżnych [11],
- współpracy sprzętowych zespołów cyfrowych [10],
- protokołów komunikacyjnych [7].

Modele sieciowe stosuje się nie tylko do badania własności systemów, ale również:

- do oceny ich wydajności [19],
- jako środka wspomagającego wykrywanie i lokalizację błędów w działających systemach o podwyższonej niezawodności [3].

Podstawowe ograniczenie zastosowań sieci Petriego wynika z faktu, że modele rzeczywistych obiektów są na ogół bardzo rozbudowane i zarówno ich poprawne konstruowanie, jak i sprawna analiza jest praktycznie możliwa tylko wtedy, gdy dostępne są odpowiednie narzędzia automatycznie wykonujące te czynności. Bardzo często narzędzia takie projektuje się z myślą o konkretnym obszarze zastosowań sieci Petriego, z przeznaczeniem dla konkretnej ich klasy, a przy tym dla rozwiązania pewnych szczegółowych problemów. Istnieją jednak również pełne systemy o całkiem ogólnych własnościach, które w szerokim zakresie umożliwiają analizę sieci używanych do różnych celów. Systemy takie powstały np. w Aarhus w Danii [20] oraz w Tuluzie we Francji [5]. Pewne prace w tym zakresie były wykonywane również w Polsce.

* * *

Mimo wyrażanych niekiedy rozczarowań sieciami Petriego (głównie w USA), ich rola w informatyce wzrasta, a także poszerza się zakres ich zastosowań. Rośnie także liczba dostępnych na rynku komputerowych systemów

automatycznego tworzenia i analizy modeli sieciowych. Powstaje coraz więcej obszernych opracowań rozwijających teorię sieci Petriego oraz w całości poświęconych tej tematyce (np. 5, 6, 14, 18). Może warto, żeby i w języku polskim ukazała się książka o sieciach Petriego?

LITERATURA

[1] Agerwala T., Flynn M. J.: Comments on coapabilities, limitations and „correctness" of Petri nets. Proc. of the 1st Annual Symposium on Computer Architecture, Gainesville, USA, pp. 81—86, 1973

[2] Andre C.: The behaviour of a Petri net on a substet of transitions. RAIRO Automatique/Systems Analysis and Control. Vol. 17, No. 1, pp. 5—21, 1983

[3] Ayache J. M., Azema P., Diaz M.: Observer — a concept for on-line detection of control errors in concurrent systems. Proc. of FTCS-9, Madison, pp. 79—86, 1979

[4] Berthomieu B.: Analyse structurelle des reseaux de Petri. Methodes et outils. These de docteur ing., Univ. de Toulouse, 1979

[5] Brams G. W.: Reseaux de Petri: theorie et pratique. Masson, Paris, 1983

[6] Brauer W. ed.: Net theory and Applications. Proc. of Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979; także w: Lecture Notes in Computer Science, No 84, Springer Verlag, 1980

[7] Diaz M.: Modelling and analysis of communication and cooperation protocols using Petri nets based models. W: Sunshine C. ed.: Protocol Specification, Testing and Verification. North-Holland Pub. Co. pp. 465—510, 1982

[8] Genrich H. J., Lautenbach K.: System modelling with high-level Petri nets. Theoretical Computer Science, Vol. 13, pp. 109—136, 1981

[9] Genrich H. J., Stankiewicz-Wiechno E.: A dictionary of some basic notions of net theory. W: [6]

[10] Gondzio M., Budkowski S.: Modelling and verifying asynchronous cooperation between microprogrammed units. Proc. of the 13th Fault-Tolerant Computing Symposium, Milano, pp. 74—77, 1983

[11] Iszkowski W., Maniecki M.: Programowanie współbieżne. WNT, Warszawa, 1982

[12] Keller R. M.: Generalized Petri nets as models for system verification. Proc. of the Conference on Petri Nets and Related Methods, Springer Lecture Notes

[13] Moella M., Pulou J., Sifakis J.: Reseaux de Petri synchronises. Res. Rep. 80, Univ. Scientifique et Medical de Grenoble, 1977

[14] Peterson J. L.: Petri Nets Theory and the Modelling of Systems, Prentice Hall, 1981

[15] Petri C. A. Fundamentals of a theory of asynchronous information flow. Proc. of the IFIP Congress, Amsterdam, pp. 386—390, 1962

[16] Petri C. A.: Introduction to general net theory. W: [6]

[17] Ramchandani C.: Analysis of asynchronous concurrent systems by timed Petri nets. MAC-TR 120, Project MAC-MIT, 1974

[18] Resig W.: Petrinetze. Eine Einfuhrung. Springer Verlag, 1982

[19] Zuberek W.: Analiza efektywności jednostek centralnych maszyn cyfrowych oraz niektóre metody jej zwiększania. Prace IINTE, Nr 26, 1979

[20] Jensen K., Hubber P., Larsen M. N., Martinsen I. M.: Petri Net Package User's Manual. DAIMI MD-46, Aarhus University, 1983

[21] Розенблюм Л.; Сети Петри. Техническая Кибернетика, № 5, стр. 12—40, 1983.

ROMAN GRABOWICZ
JANUSZ ZALEWSKI
Warszawa

System operacyjny PC-DOS (5)

W obecnym, ostatnim odcinku ogólnego opisu systemu PS-DOS omówiono krótko związane z nim oprogramowanie podstawowe, służące do przygotowywania programów użytkowych, tj. edytor (**EDLIN**), makroasembler (**MASM**), konsolidator (**LINK**) i program uruchomieniowy (**DEBUG**). Celem tego opisu nie jest jednak dokładne przedstawienie wymienionych programów, lecz zarysowanie charakterystycznych cech minimalnej liczby narzędzi programowych, niezbędnych w typowym cyklu wytwarzania oprogramowania użytkowego.

EDLIN

EDLIN służy do przygotowywania plików tekstowych, np. tekstów programów. Bardziej doświadczeni programiści twierdzą, że nie zdarzyło im się używać gorszego edytora niż **EDLIN** i chyba jest to prawdą. Ma on jednak tę zaletę, że zajmuje bardzo mało miejsca w pamięci i jest dostarczany stadardowo wraz z systemem operacyjnym, a więc jest dostatecznie rozpowszechniony i powinien być dostępny w każdej instalacji.

Edytor jest podstawowym narzędziem pracy niemal dla wszystkich osób wykorzystujących komputery. Dlatego jest naturalne, że każdy musi lepiej lub gorzej opanować technikę posługiwania się nim i większość czytelników z pewnością ją posiada. Z tego względu w niniejszym artykule niecelowy byłby dokładny opis poszczególnych poleceń i sposobów ich użycia. Tak więc poprzestano na wyliczeniu ważniejszych poleceń edytora **EDLIN** (tab. 1) bez podawania przykładów ich użycia.

Wywołanie edytora ma postać **EDLIN nazwa_pliku**, np.:

A > EDLIN TEST.ASM

Edytor zgłasza gotowość do pracy zawsze gwiazdką:

*

Jednakże sama gwiazdka pojawia się na ekranie tylko wtedy, gdy podany plik istnieje i nie mieści się w całości w buforze edytora (do bufora jest wówczas wczytywana tylko część pliku). Jeśli żądanego pliku nie ma na dysku, to przed gwiazdką wyświetlany jest meldunek:

New File
*

oznaczający, że edytor przystępuje do tworzenia nowego pliku. Jeśli podany plik istnieje i mieści się w całości w buforze edytora, to przed gwiazdką wyświetlany jest meldunek:

End of input file
*

Jeśli plik już istnieje i nie mieści się w całości w buforze edytora, to można go wczytywać częściami przesyłając z bufora na dysk zbędne segmenty **poleceniem W** (ang. write, zapisz) i wczytując nowe **poleceniem A** (ang. add, dodaj). Tekst wyprowadzany z edytora, choćby był bez zmian, jest oczywiście zapisywany na innym pliku. Ten nowy plik jest automatycznie kasowany, jeżeli praca z edytorem zostanie zakończona **poleceniem Q** (ang. quit). Jeżeli natomiast praca z edytorem zostanie zakończona **poleceniem E**, to nowy plik przyjmuje nazwę starego, a stary plik zachowuje poprzednią nazwę z nowym rozszerzeniem **BAK** (ang. backup, rezerwowy).

Podobnie jak inne edytory, **EDLIN** ma dwa tryby pracy: rozkazowy i tekstowy. Interesujące jest jednak to,

że tryb tekstowy, do którego wchodzi się wydając **polecenie I** (ang. insert, wstaw tekst) lub podając numer linii n, jest specjalnie wyróżniony na ekranie kilkuznakowym wcięciem wiersza (ang. indentation) i numerem linii, tzn.

n:*

Powrót do trybu rozkazowego następuje po przyciśnięciu klawisza **CTRL-BREAK**. Pozostałe polecenia edytora omówiono w tabeli 1.

Tabela 1. Polecenia edytora EDLIN (nawiasy kwadratowe oznaczają nieobowiązkowe części poleceń)

Postać	Znaczenie	Opis
n	Numer linii (liczba)	Poprawianie tekstu w linii n za pomocą klawiszy F1-F5, ESC, INS, DEL i ^; ostatnią linię pliku oznacza się znakiem #
[n]A	Dopisz (ang. add, append)	Wprowadzanie n nowych linii tekstu do bufora
[n], [m], d, [c] C	Kopiuj (ang. copy)	Przepisanie linii od n do m przed linię d i powielenie ich c-krotne
[n] [,m] D	Usuń (ang. delete)	Usuwanie linii od n do m
E	Wyjdź (ang. exit)	Zakończenie pracy z edytorem i powrót do systemu operacyjnego po zapamiętaniu wprowadzonych zmian
[n]I	Wstaw (ang. insert)	Wprowadzanie nowego tekstu przed linię n; wyjście z trybu I klawiszem CTRL-BREAK
[n] [,m]L	Wyświetl (ang. list)	Wyświetlenie na ekranie linii od n do m bez zmiany położenia wskaźnika linii bieżącej
[n], [m], dM	Przesuń (ang. move)	Przemieszczenie linii od n do m na miejsce znajdujące się przed linią d
[n] [,m] P	Wyświetl stronę (ang. page)	Wyświetlenie na ekranie strony tekstu od linii n do m; położenie wskaźnika linii bieżącej zmienia się na wartość m
Q	Zakończ (ang. quit)	Zakończenie pracy z edytorem i powrót do systemu operacyjnego bez zapamiętania wprowadzonych zmian
[n] [,m] [?] R [napis 1<F6> napis2]	Zastąp (ang. replace)	Wyszukanie napisu 1 między liniami n i m, i zastąpienie go napisem 2; użycie znaku zapytania powoduje wstrzymanie poszukiwania po każdorazowym znalezieniu napisu 1
[n] [,m] [?] Snapis	Wyszukaj (ang. search)	Wyszukiwanie napisu między liniami n i m; użycie znaku zapytania powoduje wstrzymanie wyszukiwania po każdym znalezieniu napisu
[n] Tplik	Prześlij (ang. transfer)	Przepisanie treści wyspecyfikowanego pliku przed linię n pliku aktualnie redagowanego
[n]W	Zapisz (ang. write)	Przepisanie n linii aktualnie redagowanego pliku na dysk

MASM

MASM jest dwuprzebiegowym makroasemblerem przeznaczonym dla komputerów IBM PC, opracowanym przez firmę Microsoft. Inny asembler tej firmy, ASM, zajmuje co prawda mniej pamięci operacyjnej, lecz nie ma wszystkich pożądanych właściwości, m. in. nie zapewnia tworzenia makroinstrukcji i nie podaje pełnych meldunków o błędach.

Makroasembler **MASM** jest programem, do którego efektywnego użycia trzeba znać język asemblerowy mikroprocesora 8080. Oczywiście w tak krótkim artykule nie można omówić listy rozkazów tego mikroprocesora. Należy zapoznać się z nią oddzielnie.

Trudno byłoby też przedstawić wyczerpująco wszystkie konstrukcje makroasemblera **MASM**. Można je jedynie wymienić. Do elementów danych języka tego asemblera należą: stałe, etykiety i zmienne (charakteryzujące się określonymi atrybutami). Natomiast wśród operatorów, oprócz typowych operatorów arytmetycznych, logicznych i relacyjnych, wyróżnia się następujące:

- przebijania atrybutów argumentu (ang. override operand's attributes),
- udostępniania wartości atrybutu argumentu (ang. return the values of attributes),
- wyodrębniania pól rekordów (ang. isolate record fields).

Przy użyciu wszystkich wymienionych operatorów można tworzyć wyrażenia. Oprócz operatorów, w języku asemblera **MASM** występuje kilka rodzajów dyrektyw (ang. pseudo-operations): dyrektywy danych (26, najliczniejsza grupa), dyrektywy warunkowe (12), dyrektywy makro (8), dyrektywy wydruku (14) i dyrektywy tzw. bloków warunkowych fałszywości, ang. false conditional blocks (3).

Ponieważ celem tak krótkiego artykułu nie może być przedstawienie, jak operować językiem asemblera **MASM**, poniżej naszkicowano jedynie sposób posługiwania się samym makroasemblerem.

Makroasembler tłumaczy program źródłowy (typu ASM), zapisany w języku asemblerowym, na program wynikowy (typu OBJ), tworząc przy tym — w razie potrzeby — dwa inne pliki: wydruk programu źródłowego (typu LST) i tablicę odwołań zewnętrznych (typu CREF, ang. cross reference). W pierwszym przebiegu następuje zdefiniowanie względnych adresów dla każdej linii kodu źródłowego, a w drugim przebiegu — wygenerowanie programu wynikowego, wydruku i pliku odwołań zewnętrznych.

Użycie makroasemblera ma postać:

MASM [progr_źródł [, [progr_wynik]] [, [wydruk]] [, [tabl_odwołań]] [/param] [;]]

gdzie zmienna param oznacza opcje makroasemblera opisane m. in. przez następujące parametry:

/D — wytworzenie wydruków po obu przebiegach asemblera

/O — przedstawienie przetłumaczonego kodu w zapisie ósemkowym.

Jeżeli w poleceniu nie występują jako argumenty nazwy wydruku i tablicy odwołań, to odpowiednie pliki nie są tworzone. Brak nazwy programu wynikowego powoduje przyjęcie dla niego nazwy domyślnej progr_źródł.OBJ. Niezakończenie polecenia średnikiem powoduje wyświetlenie zaproszenia do wprowadzenia brakujących nazw plików, dla których w poleceniu wystąpił przecinek, np. po poleceniu:

A > MASM TEST

wywietlane jest zaproszenie:

Object filename [TEST.OBJ]:

na które można odpowiedzieć dwojako — podając wprost nazwę pliku wynikowego lub przyciskając klawisz **ENTER** (wtedy plik wynikowy otrzyma nazwę domyślną podaną w nawiasach kwadratowych). Po wprowadzeniu odpowiedzi z klawiatury wyświetli się następne zaproszenie do wprowadzenia nazwy pliku wydruku:

Source listing [NUL.LST]:

Jeżeli nie poda się nazwy pliku, lecz przyciśnie klawisz **ENTER**, to wydruk zostanie wyprowadzony na plik domyślny **NUL**. Podobnie będzie w przypadku tablicy odwołań, jeżeli przyciśnie się klawisz **ENTER** w odpowiedzi na kolejne zaproszenie:

Cross reference [NUL.CRF]:

Pełna sekwencja asemblacji programu TEST mogłaby też mieć inną postać, np.:

A > MASM

Source filename [.ASM]:TEST
Object filename [TEST.OBJ]:
Source listing [NUL.LST]:
Cross reference [NUL.CRF]:

Jeżeli niepotrzebny jest wydruk i tablica odwołań, to polecenie asemblacji należy zakończyć średnikiem, tj.:

A > MASM TEST;

Tablica odwołań zewnętrznych zapisana na pliku typu CRF może zostać przetworzona za pomocą programu CREF, w celu alfabetycznego uporządkowania symboli wraz z listą ich odwołań w programie użytkowym (określonych przez numery linii), co ułatwia późniejsze uruchamianie tego programu.

LINK

LINK jest dość typowym konsolidatorem, tj. programem łączącym oddzielnie przetłumaczone moduły wynikowe i tworzącym jeden moduł ładowalny. Z modułami wynikowymi można także łączyć biblioteki, a dodatkowym produktem konsolidacji jest mapa ładowania (ang. load map).

Wywołanie konsolidatora ma postać:

LINK [moduły_wynik [, [moduł_ładow]] [, [mapa_ładow]] [, [biblioteki]] [/param] [;]]

Moduły wynikowe mogą być oddzielone znakiem „+" lub spacją i mogą mieć domyślne rozszerzenie nazwy **OBJ**, a moduły biblioteczne — rozszerzenie **LIB**. Nazwa modułu ładowalnego otrzymuje rozszerzenie **EXE**, a mapy ładowania — rozszerzenie **MAP**.

W wywołaniu konsolidatora można użyć następujących opcji:

/D — dynamiczny przydział pamięci (ang. dynamic space allocation), istotny np. w przypadku wystąpienia w programie zmiennych wskaźnikowych (ang. pointer)

/H — ładowanie programu w górny obszar pamięci (ang. high); standardowo program jest ładowany do obszaru o małych adresach

/L — utworzenie mapy ładowania z numerami i adresami (ang. line numbers) instrukcji modułów wynikowych

/M — utworzenie alfabetycznej listy (ang. map) symboli globalnych, zdefiniowanych w module ładowalnym (wartość, segment i adres względny)

/N — dołączenie biblioteki domyślnej (ang. no default library search) o nazwie zgodnej z konsolidowanym oprogramowaniem

/P — wstrzymanie przebiegu łączenia (ang. pause), np. w celu wymiany dyskietki

/S:rozmiar — przydział stosu (ang. stack), o standardowej wielkości równej 512 B.

W poleceniu **LINK** można — oczywiście — nie podawać żadnych argumentów. Wtedy zostaną wyświetlone zaproszenia do podania identyfikatorów kolejnych plików, np. po poleceniu:

LINK

dialog z komputerem może wyglądać następująco:
Object Modules [.OBJ]: TEST+TEKST M
Run File [TEST.EXE]: TEST1
List File [NUL.MAP]: TEST
Libraries [.LIB]: BIBLIA

Aby uniknąć każdorazowego wprowadzania tych samych odpowiedzi przy powtarzalnym wywoływaniu konsolidatora, można je zapisać na pliku typu **BAT** (np. TEST.BAT) i dokonać konsolidacji używając polecenia:

LINK @TEST.BAT

Utworzony przez konsolidator program ładowalny typu EXE nie jest jeszcze w pełni gotowy do wykonania (choć może być wykonany). W czasie ładowania go do pamięci system operacyjny musi wykonać pewne dodatkowe operacje, związane z położeniem programu w pamięci, jego rozmiarem, utworzeniem stosu itp. Do wcześniejszego przekształcenia programu ładowalnego w wykonywalny obraz typu **COM** można użyć polecenia (programu usługowego) **EXE2BIN** (ang. EXE to binary). Postać typu **COM** jest bardziej zwarta (o kilkaset bajtów krótsza). Nie wszystkie programy można jednak przekształcić przed załadowaniem na postać typu **COM**. Programy napisane w większości języków wysokiego poziomu, na ogół, muszą być ładowane w postaci typu **EXE**.

Każdy wykonywalny program musi zawierać specjalny blok sterujący, utworzony prze system operacyjny, tzw. prefiks segmentu programu (ang. program segment prefix, PSP), zajmujący 256 bajtów (100H), począwszy od adresu względnego 0, w segmencie przydzielonym programowi. Służy on do wymiany informacji między programem a systemem PC-DOS. Po prefiksie, tzn. od adresu 100H, jest umieszczony sam segment kodu programu (ang. code segment), następnie segment danych (zawierający wszystkie zmienne) i segment stosu. Prefiks PSP nie istnieje na dysku, lecz jest każdorazowo tworzony przez system operacyjny podczas ładowania programu.

DEBUG

DEBUG jest programem uruchomieniowym, którego zasadniczym zadaniem jest wspomaganie programistów w usuwaniu defektów z programów użytkowych. Kilkanaście typowych poleceń tego narzędzia programowego przedstawiono w tabeli 2. Umożliwiają one m. in. krokowe wykonywanie programu użytkowego i oglądanie w tym czasie jego stanu oraz zmian tego stanu.

Tabela 2. Skrócony opis poleceń programu DEBUG (nawiasy kwadratowe oznaczają nieobowiązkowe części poleceń)

Postać	Znaczenie	Opis
A [adres]	Asembluj (ang. assemble)	Bezpośrednie wprowadzanie do pamięci instrukcji języka asemblera; ich liczba jest ograniczona do 200
C zakres adres	Porównaj (ang. compare)	Porównanie zawartości obszaru pamięci zdefiniowanego przez argument zakres z obszarem rozpoczynającym się od argumentu adres
D [adres]	Składuj (ang. dump)	Wyświetlanie zawartości obszaru pamięci
E adres [zawartość]	Wprowadź (ang. enter)	Modyfikowanie zawartości pamięci
F zakres [zawartość]	Wypełnij (ang. fill)	Wypełnianie obszaru pamięci podaną zawartością
G [=adres] [adres [adres ...]]	Wykonaj (ang. go)	Wykonanie uruchamianego programu od podanego adresu i realizacja punktów wstrzymania w miejscach określonych przez następne adresy
H wartość wartość	Oblicz szesnastkowo (ang. hexaritmetic)	Obliczanie sumy i różnicy dwóch liczb szesnastkowych
I adres	Wprowadź (ang. input)	Wprowadzanie zawartości portu zdefiniowanego przez adres
L [adres [dysk sektor sektor]]	Załaduj (ang. load)	Ładowanie pliku dyskowego lub zawartości określonych sektorów dyskowych (maksymalnie 80H); pliki typu COM i EXE są ładowane zawsze pod adres CS:100H
M zakres adres	Przesuń (ang. move)	Przesunięcie zawartości obszaru określonego przez argument zakres do obszaru rozpoczynającego się od argumentu adres
N nazwa_ pliku	Nazwij (ang. name)	Nazwanie tworzonego lub wczytywanego pliku (programu)
Q	Zakończ (ang. quit)	Zakończenie działania programu DEBUG i powrót do systemu operacyjnego bez zarejestrowania na dysku wprowadzonych zmian
R [rejestr]	Rejestry (ang. registers)	Wyświetlanie zawartości rejestrów
S zakres zawartość	Przeszukaj (ang. search)	Przeszukanie obszaru pamięci określonego przez zakres ze względu na określoną zawartość
T [adres] [wartość]	Prześledź (ang. trace)	Wykonanie określonej liczby kolejnych instrukcji i wyświetlenie stanu rejestrów procesora
U [adres]	Disasembluj (ang. unassemble)	Przekształcenie zawartości ustalonego obszaru pamięci (16 lub 32 bajty) na instrukcje w języku asemblera
W [adres [dysk sektor sektor]]	Zapisz (ang. write)	Zapisanie na dysk określonego pliku lub obszaru pamięci

Typowe użycie programu **DEBUG** polega na wykonaniu modyfikacji zawartości określonej komórki pamięci (instrukcji lub danych programu). Wywołanie programu

DEBUG i załadowanie programu uruchamianego może mieć postać bezpośrednią:

```
A > DEBUG TEST.COM
```

lub alternatywną (myślnik jest znakiem gotowości):

```
A > DEBUG
—N TEST.EXE
—L
```

Jeżeli program (lub plik tekstowy) jest załadowany, to można go obejrzeć używając polecenia składowania, np.:

```
—D
08F1:0100  xx yy zz ...  (16 znaków tekstu)
08F1:0110  xx yy zz ...  (16 znaków tekstu)
```

itd., lub polecenia disasemblacji, np.:

```
—U 24A
0A7E:034A  B402      MOV  AH,02
0A7E:034C  BA7F00    MOV  DX,007F
0A7E:034F  42        INC  DX
itd.
```

Warto zauważyć, że do adresu podanego w obu poleceniach dodane zostało wyrównanie (ang. offset) 100H, ponieważ kod programu zaczyna się od takiego adresu względnego (jest poprzedzony prefiksem PSP).

Można obejrzeć także zawartość rejestrów, np. rejestru CX zawierającego rozmiar pliku, np.:

```
—R CX
CX xxxx
```

Wpisania nowych rozkazów, znaków, wartości itp. można dokonać poleceniem wprowadzania, np.:

```
E 34D FF FF
```

lub poleceniem asemblacji, np.:

```
A 34C MOV DX,FFFF
```

Przed zakończeniem pracy należy przepisać poprawiony program na dysk poleceniem **W**.

Warto dodać, że pliki typu **EXE** są traktowane przez **DEBUG** nieco inaczej niż wszystkie inne. Dlatego, w celu przeprowadzenia zwykłych operacji na pliku tego rodzaju, należy go przedtem przemianować, zmieniając rozszerzenie nazwy.

* * *

Użycie programów usługowych, przedstawionych w obecnym odcinku opisu systemu PC-DOS, jest niezbędne w procesie opracowywania oprogramowania użytkowego. Dlatego też opanowanie techniki posługiwania się tymi programami, tu zaledwie zarysowanej, ma istotne znaczenie dla każdego programisty i wymaga samodzielnej pracy z komputerem przy użyciu odpowiednich podręczników firmowych.

MAREK PAWŁOWSKI
Instytut Informatyki
Politechniki Warszawskiej

Programator pamięci UVEPROM serii 27XXX

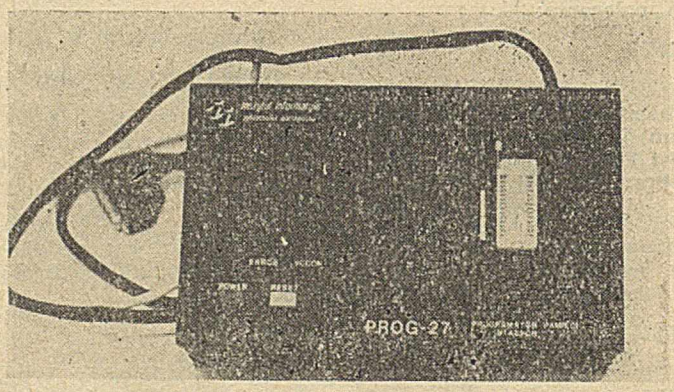
Część programów w systemach mikroprocesorowych — fragmenty systemu operacyjnego, translatory — jest przechowywana w pamięci stałej. W wielu wypadkach występuje konieczność zmiany firmowej zawartości pamięci stałej. Do tego celu konieczne jest dodatkowe urządzenie — programator pamięci stałych.

W Instytucie Informatyki Politechniki Warszawskiej opracowano programator PROG27, przeznaczony do programowania pamięci UVEPROM firmy Intel następujących typów: 2716, 2732, 2732A, 2764, 2764A, 27128, 27256, 27512 oraz ich odpowiedników innych firm (m.in. K573PΦ2, K573PΦ5).

PROG27 jest zrealizowany w technice MSI TTL i wymaga zewnętrznego sterowania z systemu mikroprocesorowego. W tym celu został wyposażony w łącze szeregowe V24 (styk S2), działające w trybie asynchronicznym z bitem startu, ośmioma bitami danych, bitem parzystości i dwoma bitami stopu. Szybkość transmisji została określona na 19 200 bodów. Sygnały łącza V24 są wyprowadzane na wtyk ELTRA 871025, zgodnie z normą PN-75/T05052.

Przez łącze V24 przesyłane są do programatora PROG27 polecenia włączenia zasilania, odczytu stanu zasilania, zapisu rejestru adresowego (polecenie 4-bajtowe), zapisu rejestru danych do programowania (polecenie 3-bajtowe), inicjowanie odczytu bajtu z pamięci EPROM itp. Tym samym łączem odbierana jest informacja wysyłana przez programator na polecenia z mikroprocesorowego systemu sterującego.

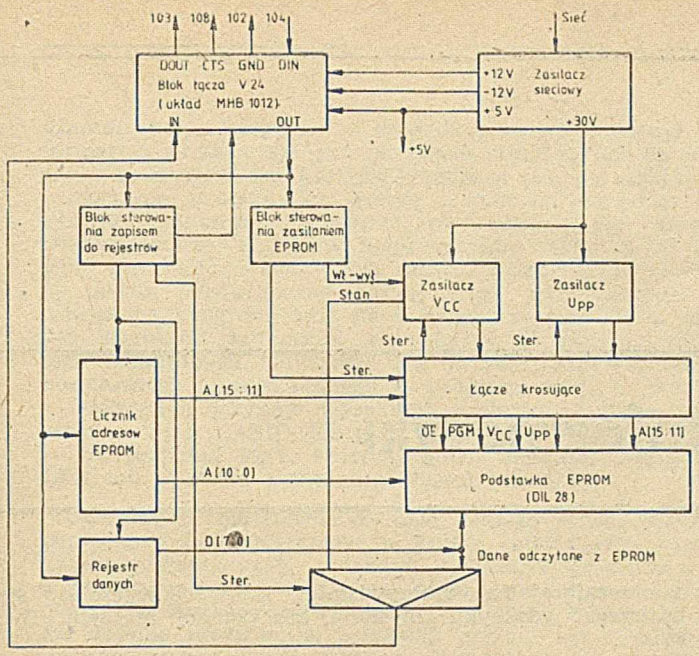
Uproszczony schemat blokowy programatora przedstawiono na rysunku 1. Współpracę z otoczeniem umożliwia



Programator pamięci stałych PROG27

blok łącza V24 zawierający układ MHB1012. Układ ten jest programowany sprzętowo (przez odpowiednie połączenia) i bezpośrednio po włączeniu zasilania jest gotowy do pracy.

Kolejne bajty prawidłowo odczytanych danych są interpretowane przez programator jako polecenia, zapisywane do rejestrów w blokach sterujących, skąd wpływają na pracę programatora. Zapis polecenia do bloku sterowania zapisem rejestrów powoduje zinterpretowanie bajtów, występujących po poleceniu jako danych zapisywanych do wybranych rejestrów: danej do programowania, bardziej i mniej znaczącego bajtu adresu. W wypadku błędnej transmisji (zła parzystość) odczytane dane są ignorowane i ustawiany jest przerzutnik błędu. Stan tego przerzutnika może być odczytany za pośrednictwem łącza V24.

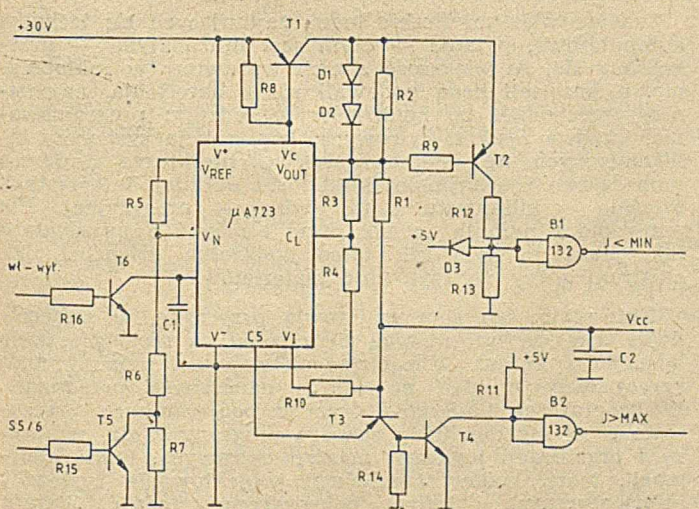


Rys. 1. Schemat blokowy programatora PROG27

ZASILACZ PAMIĘCI PROGRAMOWANYCH

PROG27 został wyposażony w programowane źródła napięciowe określające — dla danego typu pamięci UVEPROM podczas jej programowania — odpowiednie parametry zasilania. Zasilacz umożliwia podanie na programowaną pamięć napięcia V_{CC}= +5,0 V lub V_{CC}= +6,0 V z ograniczeniem prądowym ustawionym na 130 mA. Zasilacz V_{CC} zawiera układy umożliwiające sprawdzenie wartości prądu I_{CC}, pobieranego przez pamięć EPROM. Schemat tego zasilacza przedstawiono na rysunku 2. Gdy pamięć EPROM pobiera zbyt mały prąd (I_{CC}<20 mA), wtedy prąd ten płynie przez rezystor R2 powodując na nim spadek napięcia mniejszy od 1,4 V. Powoduje to zatkanie tranzystora T2 i tym samym na wyjściu bramki B1 wystąpi stan jedynki logicznej, sygnalizując brak lub błędne włożenie pamięci w podstawkę EPROM. Gdy I_{CC}>130 mA, wtedy zadziała ograniczenie prądowe układu μA723 i tranzystor T3 zacznie przewodzić, odblokowując T4, co z kolei spowoduje pojawienie się jedynki logicznej na wyjściu bramki B2. Stan zasilania może być odczytany odpowiednim poleceniem przez łącze V24. Tranzystor T5, sterując dzielnikiem napięcia odniesienia R5-R6 (gdy T5 jest nasycony, S5/6=1) lub R5—(R6+R7), wyznacza poziom napięcia wyjściowego zasilacza. Stan zera logicznego na wejściu wł-wył uruchamia zasilacz, a jedynki logicznej — wyłącza go.

Zasilacze o podobnej budowie są stosowane w programatorze PROG2 (por. Informatyka, nr 6 1984). Wykorzystując możliwości zasilacza V_{CC} przez odpowiednią obsługę programową programatora PROG27, można zabezpieczyć



Rys. 2. Schemat ideowy zasilacza V_{CC}

układy UVEPROM przed zniszczeniem po nieprawidłowym włożeniu ich do podstawki (po wykryciu stanu I_{CC}>130 mA następuje wyłączenie zasilacza).

Na płycie czołowej programatora PROG27 znajduje się 28-końcówkowa zamykana podstawka dla wszystkich wymienionych typów pamięci UVEPROM. Znaczenie jej końcówek jest zmieniane w zależności od typu programowanej pamięci przy użyciu konfiguratora, którego wtyk szufladowy ELTRA 871050 znajduje się również na płycie czołowej. Każdy typ programowanej pamięci wymaga innego konfiguratora, który poza dostarczeniem do podstawki odpowiednich sygnałów, zawiera również elementy określające poziom napięcia V_{PP}, czas trwania impulsu programującego oraz typ pamięci, dla której jest przeznaczony. Dzięki takiemu rozwiązaniu przystosowanie PROG27 do programowania nowego typu pamięci UVEPROM jest bardzo proste.

OPROGRAMOWANIE

Programator PROG27 wymaga odpowiedniej obsługi programowej ze strony systemu cyfrowego wyposażonego w łącze V24. Obsługę taką zapewnia program PROG napisany w języku asemblera mikroprocesora 8080, a działający pod nadzorem systemu operacyjnego CPM 2.2. Dla poprawienia działania programu PROG, system mikroprocesorowy powinien zawierać: jednostki dyskowe obsługiwane przez system CP/M, blok DMA, blok V24 z układem 8251 i generatorem zegara transmisyjnego, zapewniającym szybkość przesyłania 19 200 bodów. Program obsługi programatora PROG27 umożliwia:

- określenie typu programowanej pamięci
- odczyt do bufora pliku typu HEX lub COM
- zapis zawartości bufora do pliku dyskowego HEX lub COM
- wyświetlenie wskazanego obszaru bufora
- modyfikację zawartości bufora
- określenie granic obsługiwanego obszaru bufora
- określenie numeru obsługiwanej kostki EPROM
- odczyt zawartości pamięci EPROM do fragmentu bufora odpowiadającego danej kostce pamięci
- weryfikację zawartości pamięci EPROM z buforem lub wzorcem 0FFH
- programowanie pamięci EPROM zawartością bufora.

W trakcie działania programu PROG na ekranie monitora systemowego wyświetlane jest menu zawierające zestaw funkcji programu oraz bieżący stan programu, określający typ programowanej pamięci, nazwę aktywnego pliku dyskowego, granice obszaru pamięci buforowej, adres bazowy pliku, aktualny oraz maksymalny numer kostki EPROM. W menu wyświetlane są również granice obszaru bufora, będącego obrazem aktualnej kostki programowanej pamięci. Po wybraniu funkcji, na ekranie monitora pojawiają się pytania o parametry jej wykonania oraz komunikaty o ewentualnych błędach.

Realizację funkcji związanych z dostępem do pamięci stałej umieszczonej w podstawce programatora rozpoczyna transmisja poleceń sprawdzenia poprawności transmisji (po każdej transmisji odczytywany jest stan przerzutnika błędu transmisji). Następnie odczytywany jest kod konfiguratora, który zostaje porównany z zadeklarowanym typem pamięci. Kolejną czynnością jest załączenie zasilacza V_{CC} i sprawdzenie jego stanu. Po sprawdzeniu poprawności działania układu następuje przejście do realizacji właściwej funkcji, odczytu lub programowania pamięci UVEPROM. Powyższe zabezpieczenie ma na celu uzyskanie maksymalnej niezawodności działania programatora PROG27.

Program PROG oraz wykorzystanie właściwości programatora PROG27 pozwala na programowanie pamięci o pojemności większej od 4 KB szybką procedurą programowania inteligentnego. Zmniejsza to czas programowania pamięci do jednej minuty dla każdych 2 KB organizacji EPROM. Przykładowo, czas programowania pamięci 27256 wynosi około 17 minut. Stosując procedurę standardową czas ten wyniósłby około 32 minut.

* * *

Programator PROG27 został wdrożony do produkcji w Zakładzie Doświadczalnym Instytutu Informatyki Politechniki Warszawskiej. Dodatkowe informacje na jego temat można otrzymać telefonicznie pod numerem 21007-811 w Warszawie.

RYSZARD TADEUSIEWICZ
Instytut Informatyki
Akademia Górniczo-Hutnicza
Kraków

Nauczanie informatyki

Komputery tak dalece przeniknęły do nauki, techniki i gospodarki, że ich wykorzystanie stało się koniecznością. Jednak nie zawsze jest tak rzeczywiście, gdyż często mamy tu do czynienia z fascynacją, kiedy specjalista z dowolnej dziedziny, jedynie chwilowo zajmujący się informatyką dla rozwiązania pewnego konkretnego zadania, daje się wciągnąć w nie mającą końca pogoń za mirażem, jakim jest doskonały program komputerowy. Narzędzie zaczyna przesłaniać cel, aż wreszcie samo staje się celem.

EPIDEMIA INFORMATYKI

Ośrodki obliczeniowe (głównie uczelnie, ale obawiam się, że nie tylko) stają się niemal wyłącznym polem działania licznych „naukowców hybrydowych" — mechaników-informatyków, lekarzy-informatyków, językoznawców-informatyków itp. Zamiast myśleć — czeka się na wyniki, zamiast konstruować stanowiska pomiarowe — konstruuje się programy, zamiast wykonywać doświadczenia — nabywa się doświadczeń w programowaniu. Codziennie komputery zadrukowują setki stronic papieru, pożerają setki tysięcy kart dziurkowanych, obliczają i podają do wiadomości miliony liczb, wykonują setki miliardów operacji matematycznych.

Nowy wymiar tego problemu wiąże się z lawiną mikrokomputerów: „domowych", „osobistych", „biurowych" itp. Masowo kupowane, stały się dumą i fetyszem swych właścicieli, symbolem prestiżu i nowoczesności. Wykorzystywane w rzeczywistości do różnych celów (najczęściej do gier...), nabywane są zwykle z myślą o zastosowaniach zawodowych. Z pomocą takiego domowego komputera można — nie ruszając się z domu, za dnia i w nocy, w dzień powszedni i święto — tworzyć algorytmy, pisać programy i liczyć, liczyć, liczyć...

Czy zawsze te obliczenia mają sens? Czy wszystkie, tysiącami drukowane liczby stają się podstawą głębokich przemyśleń? Jak często wydruki z komputera służą jako parawan dla przedmiotowej ignorancji i braku koncepcji? Ile spośród licznych błyskotliwych zastosowań komputera ma charakter istotne potrzeby, a ile wynika z mody na pseudonowoczesność?

Na każde z tych pytań udzielenie odpowiedzi jest bardzo trudne. Ogromna liczba osiągnięć współczesnej nauki i techniki jest bezpośrednim następstwem szerokiego wykorzystania komputerów, dlatego musimy upowszechniać wiedzę informatyczną wśród nieprofesjonalistów. Niewątpliwie komputer dla każdego wykształconego człowieka stanowi źródło nieustannego wyzwania, porównywalnego jedynie z intelektualną atrakcyjnością szachów, stąd łatwość ulegania wskazanej wyżej fascynacji, ze wszystkimi jej ujemnymi następstwami. Jest wielce prawdopodobne, że wynalazek komputera będzie miał podobne skutki, jak wynalazek druku, który spełniając ogólnie znane kulturotwórcze funkcje, spowodował zalew informacji pisanej, z którą obecnie nie potrafimy się uporać, i który powoduje, że na przykład ten artykuł prawie na pewno nie dotrze do tych czytelników, do których w intencji autora jest adresowany. Być może, pośród milionów liczb produkowanych przez komputery będziemy jeszcze mniej wiedzieli o otaczającej nas rzeczywistości, którą odzwyczaimy się postrzegać własnymi zmysłami, bez pomocy maszyny. Być może...

Jednak zanim nie jest jeszcze za późno, możemy przynajmniej starać się spodziewanemu złu zaradzić. Najbardziej celowe wydaje się tu działanie w zakresie nauczania podstaw informatyki, szczególnie w odniesieniu do nauczania tego przedmiotu w szkołach średnich oraz na kierunkach studiów nie związanych bezpośrednio ze specjalnością Informatyka.

STAN NAUCZANIA INFORMATYKI

Trudności nauczania podstaw informatyki wynikają generalnie z trzech źródeł:

- braku jednolitej definicji zakresu przedmiotu,
- niedostatku środków technicznych,
- niskich kwalifikacji kadry nauczającej.

Jestem przekonany, że w stosunku do znaczenia każdego z tych źródeł można toczyć zacięte spory. Powołując się na istniejące i zatwierdzone programy można wykazać, że dokładnie wiadomo, czego nauczać. Wskazując na liczbę godzin przeznaczonych w uczelnianych ośrodkach obliczeniowych na cele dydaktyczne (w szczególności na wykonywanie programów studenckich) oraz wyliczając dostępne w szkołach mini- i mikrokomputery, jako niepoważne można odrzucić przypuszczenia o brakach sprzętowych. Na koniec, mając liczną i utytułowaną kadrę, nie można wątpić w najwyższy możliwy poziom nauczycieli. Czy jednak te optymistyczne opinie i oceny odpowiadają rzeczywistości? W jakim stopniu postęp, jaki obserwujemy w nauczaniu informatyki w ciągu ostatnich lat, dotyczył tego, co naprawdę ważne?

Tematyka nauczania

Zadając większej liczbie ludzi, zajmujących się techniką komputerową, pytanie — czym jest informatyka — przekonamy się, że większość z nich wprawimy w zakłopotanie, a pozostali będą podawali różne określenia, najchętniej przechodząc do konkretów w rodzaju: programowanie, budowa maszyn cyfrowych, konstrukcja systemów informacyjnych itp. Poszukiwania w literaturze dostarczą nam szybko przynajmniej kilkunastu definicji informatyki. Właśnie — kilkunastu, a nie jednej — precyzyjnej. Nie przeszkadza nam to jednak pisać i mówić o informatyce tak, jakby to była tak samo konkretna dziedzina, jak elektronika czy wytrzymałość materiałów.

Naturalnie, ten stan ma swoją przyczynę, w szczególności taką, że informatyka liczy sobie na serio nie wiele więcej niż 20 lat. Chcąc jednak nauczać, musimy mieć sprecyzowany pogląd na zakres nauczanego przedmiotu. W programach nauczania oraz w podręcznikach prezentowane są zagadnienia z zakresu programowania, struktury i organizacji logicznej maszyn cyfrowych, teorii informacji, teorii kodów, technologii projektowania systemów informatycznych i innych pokrewnych dziedzin. Wyraża się przy tym nadzieję, że suma tych zagadnień stanowi reprezentację przedmiotu informatyka. Nie ma to jednak

uzasadnienia, szczególnie w wypadku, kiedy nauczanie informatyki prowadzone jest w szkole średniej lub na kierunkach studiów, na których przedmiot ten ma znaczenie pomocnicze, uzupełniając wiedzę zawodową przyszłego inżyniera metalurga, mechanika, budowlanego itp. Liczba wiadomości, zawartych w nieprecyzyjnie rozumianym pojęciu informatyki, stoi w tak rażącej dysproporcji do liczby godzin, które można poświęcić na jej nauczanie, że wykładany program staje się żałosnym kompromisem między szerokimi zamiarami a realiami. Najczęstszy obraz tego kompromisu sprowadza się do nauczania — bardzo pospiesznego i nieporządnego — programowania w jednym z dostępnych języków wysokiego poziomu. Zwykle językiem tym jest BASIC, lub przy posiadaniu lepszego sprzętu — na przykład PL/1, Fortran, Cobol, niekiedy ASSEMBLER lub C, Algol i aktualnie Pascal lub Ada.

Uważam, że niezależnie od tego, jakiego języka uczymy, popełniamy błąd, mszczący się potem zatłoczonymi ośrodkami obliczeniowymi, zalewem kiepskich programów i wspomnianym na wstępie modelem „specjalistów hybrydowych". Można postawić tezę, że dlatego tak wielu ludzi obecnie programuje, ponieważ jedynie nieliczni potrafią naprawdę wykorzystywać komputery. Powszechnie stosowany system nauczania informatyki wiele uwagi poświęca programowaniu, zaś zbyt mało wykorzystywaniu oprogramowania bibliotecznego, umiejętności precyzyjnego formułowania problemów w postaci algorytmów (zalecanych ewentualnie później do oprogramowania specjalistom), prawie w ogóle nie mówi się o metodyce informatycznej analizy problemów z uwzględnieniem charakteru danych wejściowych i pożądanej formy wyników.

Nie potrafię naturalnie podać tematyki optymalnego modelu nauczania informatyki, jednak jestem głęboko przekonany, że konieczna jest szeroka dyskusja nad takim modelem. W szczególności jestem zdania, że mając do dyspozycji jeden semestr zajęć — jest niedopuszczalną rozrzutnością poświęcać go w całości na zmagania z kolejnymi instrukcjami i konstrukcjami języka programowania przy praktycznie całkowitym pominięciu wszelkich innych problemów informatyki. Jest znacznie bardziej celowe zapoznanie studenta z właściwościami komputera i metodologią programowania za pomocą maksymalnie prostego języka programowania. Zaoszczędzony czas może i powinien być wykorzystany przede wszystkim dla nauczania rozumienia techniki informatycznej — jej możliwości i ograniczeń.

Środki techniczne

W większości uczelni podstawowym środkiem technicznym, wykorzystywanym do nauczania informatyki, jest duży komputer. Programów studenckich jest dużo (zazwyczaj, i słusznie, żąda się, aby każdy student napisał i uruchomił przynajmniej jeden program), a jednocześnie są to programy dość krótkie (czas ich wykonania jest dodatkowo skracany wskutek nieuchronnie występujących błędów), dlatego zwykle stosuje się obsługę programów z wykorzystaniem trybu wsadowego, zwiększającego sprawność przetwarzania. Środki te są konieczne ze względu na potrzebę ograniczenia i tak wysokich kosztów obsługi programów dydaktycznych. Student jest jednak odsunięty od komputera, mając do czynienia wyłącznie z plikami kart, zawierającymi program i dane oraz z wydrukami zawierającymi wyniki lub (częściej) diagnostykę błędów.

Natomiast kontakt studenta z mikrokomputerem jest bezpośredni. Sytuacja ta nie jest pozbawiona — z dydaktycznego punktu widzenia — swoistych wad. Trudno jest przy użyciu mikrokomputera rozwiązywać duże problemy w sposób właściwy — czyli bazując na oprogramowaniu bibliotecznym. Wytwarza to omówiony odruch sięgania do samodzielnego programowania przy każdej okazji korzystania z informatyki, a ponadto uczy korzystania z języków, których przydatność dydaktyczna jest powszechnie krytykowana. Zatem zarówno duży system komputerowy, jak i sieć mikrokomputerów są rozwiązaniami dydaktycznie nieoptymalnymi. Można jednakże wskazywać na korzystne strony takiej sytuacji, polegające na przygotowaniu studenta do warunków, w jakich zazwyczaj przyjdzie mu działać po studiach.

Stwierdzenie tego faktu skłania do ogólniejszej refleksji. Otóż, generalnie w uczelni komputer jest wykorzystywany do prac naukowych, prac administracyjnych (przetwarzanie danych dla potrzeb uczelni) i do dydaktyki. Te trzy kierunki zastosowań różnią się zasadniczo wymaganiami w stosunku do konfiguracji systemu komputerowego. Tymczasem, z reguły istnieje tylko jeden system cyfrowy, gdyż koszty sprzętu informatycznego nadal są niezwykle wysokie. Charakterystyczne jest przy tym, że w tej sytuacji większość uczelni posiada sprzęt przystosowany głównie do prac naukowych (duża i szybka jednostka centralna ze skromnym wyposażeniem w urządzenia wejścia-wyjścia, zwłaszcza interakcyjne i ograniczoną liczbę pamięci masowych). To samo w znacznym stopniu dotyczy firmowego i systemowego oprogramowania, dostępnego w większości uczelni. Jest to sytuacja nieprawidłowa, gdyż analiza wykorzystania czasu maszyny dowodzi, że dydaktyka i zadania studenckie zajmują więcej czasu niż prace naukowe, przy czym te ostatnie także, przynajmniej w części, mogą być wykonywane na maszynie dostosowanej swą architekturą do zadań dydaktycznych.

Wydaje się, że najpilniejsze jest dokładne przeanalizowanie wymagań i potrzeb, jakie dydaktyka stawia sprzętowi komputerowemu, i podjęcie starań, aby kolejne instalowane w ośrodkach akademickich i w szkołach systemy spełniały te wymagania. Naturalnie, nie potrafię zaproponować tu optymalnego modelu struktury systemu cyfrowego, przeznaczonego do celów dydaktycznych, wydaje się jednak możliwe wskazanie kilku cech pożądanych dla takiego systemu. Po pierwsze, konieczne jest wyposażenie umożliwiające wielodostępną interakcyjną pracę licznych grup studenckich. Ze względu na to, że początkujący programiści popełniają wyjątkowo dużo błędów, a także dlatego, że celem pisanych i uruchamianych przez studentów programów jest nauka, a nie uzyskanie określonych wyników obliczeń, których wydruk ma być wykorzystany w charakterze dokumentu, celowe jest szerokie stosowanie monitorów ekranowych zamiast urządzeń drukujących. Rozwiązanie takie, jakkolwiek sprzętowo droższe, wydaje się ekonomicznie uzasadnione, po obliczeniu kosztów niepotrzebnie zużywanego papieru z wydrukami licznych błędnych kompilacji oraz błędnych wykonań studenckich programów. Problem liczby końcówek postulowanego systemu musi być rozważany indywidualnie, z uwzględnieniem programu studiów, liczby studentów i możliwości lokalowych uczelni. Niecelowe jest bowiem grupowanie większej liczby końcówek w jednym pomieszczeniu jako że praca z komputerem, bardziej niż jakakolwiek inna forma działalności umysłowej, wymaga skupienia, łatwego do osiągnięcia w kabinie, a trudnego w sali grupującej kilkanaście osób.

Na podkreślenie zasługuje fakt, że pomimo działania w trybie wielodostępnym, komputer obsługujący dydaktykę uczelni nie musi być zbyt szybki ani nie musi posiadać wyrafinowanego systemu operacyjnego. Tempo wprowadzania informacji nie jest zbyt duże, jako że początkujący studenci redagują swoje programy znacznie wolniej i z większą liczbą poprawek niż doświadczeni programiści. Podobnie czas reakcji systemu, choć pożądany najkrótszy, może być wydłużony w stosunku do czasu wymaganego w systemach profesjonalnych. Z tego między innymi powodu można rozważać stosowanie w dydaktyce łatwo dostępnych i tanich mikrokomputerów. Pamiętać trzeba przy tym, aby cechy właściwe tylko mikrokomputerom (np. skromne oprogramowanie, język Basic, ubogie pamięci masowe itp.) nie zdominowały w świadomości uczniów cech uniwersalnych, będących treścią nauczania.

Po drugie, celowe jest wyposażenie dydaktycznego systemu komputerowego w specjalizowane oprogramowanie służące usprawnieniu dydaktyki, ze szczególnym uwzględnieniem możliwości przekazywania komputerowi czynności kontrolnych. W zakresie nauczania podstaw informatyki istnieją ogromne możliwości zwiększenia sprawności przy równoczesnym zmniejszeniu pracochłonności. Niektóre z tych możliwości wiążą się z faktem, że w nauczaniu podstawowym zawsze następuje czas, kiedy student musi sprawdzić swoje umiejętności przez samodzielne napisanie i uruchomienie programu na zadany temat. Zadania sta-

wiane studentom dotyczą zwykle prostych zagadnień, egzemplifikujących poszczególne problemy i technik programowania, a wynik otrzymany z programu studenckiego służy jedynie dla kontroli. Wydaje się, że w tej sytuacji możliwe i celowe jest powierzenie kontroli maszynie, która może być wyposażona w zbiór (jak największy i ustawicznie przez grono nauczające poszerzany) zadań wraz z rozwiązaniami pozwalającymi na weryfikację poprawności programu studenckiego. Oczywiście, budowa takiego zbioru „etiud" programowych, a zwłaszcza zaprojektowanie skutecznych testów badających poprawność i optymalność (na przykład czas obliczeń lub liczbę instrukcji) programów studenckich wymaga sporego wysiłku i pomysłowości. Zaletą takiego systemu jest jednak pozostawienie wypróbowanych zadań w „banku danych" komputera i stałe poszerzanie ich zasobu w miarę eksploatacji systemu, a także ujednolicenie programu i wymagań dla grup studenckich prowadzonych przez różnych asystentów. Ewentualna wada, polegająca na powtarzaniu się zadań w kolejnych latach studiów i wynikająca stąd możliwość prosperowania „giełdy" studenckiej, jest możliwa do przezwyciężenia drogą poszerzania zasobów „banku zadań" (wada ta jest zresztą aktualna także i w obecnym systemie prowadzenia zajęć). Oczywiście, oprócz zadań, w skład „banku programów" mogą wchodzić programy nauczające, przykładowe, ilustracyjne czy wreszcie trenujące stopień opanowania poszczególnych umiejętności.

Trzecią, bardziej odległą potrzebą w zakresie sprzętu informatycznego, służącego dydaktyce podstaw informatyki, jest posiadanie urządzeń specjalistycznych, przeznaczonych wyłącznie do celów dydaktycznych. Jako dydaktyk, prowadząc zajęcia z zakresu elektronicznej techniki obliczeniowej, niejednokrotnie odczuwam zakłopotanie wywołane konfliktem pomiędzy potrzebą poglądowego przedstawienia określonych konstrukcji programowych lub postaci komputerowej diagnostyki błędów (do czego obecnie typowo używa się tablicy lub rzutnika pisma) a kameralnością oryginalnych dokumentów, produkowanych przez komputerowe urządzenia wyjściowe. Wydruk lub obraz na ekranie typowego monitora mogą wygodnie oglądać 2—3 osoby. Przy grupie powyżej sześciu osób jest to praktycznie niemożliwe, zwłaszcza jeśli dydaktyk musi komentować wydruk, wskazując jego poszczególne elementy.

Aktualnie trudność tę usiłuje się rozwiązać kilkoma sposobami, z których każdy ma istotne wady. Najczęściej dąży się do zmniejszenia liczebności grup studenckich. Jest to rozwiązanie z wielu punktów widzenia korzystne, pozwala między innymi zbliżyć się do ideału, jakim jest zindywidualizowane nauczanie praktyczne każdego studenta, a także ułatwia operowanie grupą studencką bezpośrednio w ośrodku obliczeniowym w razie prowadzenia zajęć przy maszynie. Jest to jednak rozwiązanie bardzo kosztowne, gdyż wymaga (w przeliczeniu na tę samą liczbę studentów) znacznie większej liczby godzin pracy nauczycieli, a także kłopotliwe w warunkach ograniczonej liczby sal ćwiczeniowych. Sytuacja mogłaby jednak ulec radykalnej poprawie, gdyby możliwe było zainstalowanie w salach ćwiczeniowych specjalizowanych urządzeń wejściowo-wyjściowych, dołączonych do użytkowanej maszyny cyfrowej. Specjalizacja wspomnianych urządzeń powinna polegać na wykorzystaniu wielkogabarytowych wyświetlaczy, zastępujących konwencjonalną tablicę. Wyświetlacz taki, oparty na technice diod LED, ciekłych kryształów lub dostatecznie jasny monitor kineskopowy i układ optyczny umożliwiający rzutowanie obrazu na ekran, może oddać nieocenione usługi i — mimo swoich kosztów — byłby opłacalny z uwagi na zmniejszenie kosztów eksploatacji. Jest to jednak rozwiązanie bardzo odległe w czasie. Alternatywą może być sieć sprzężonych mikrokomputerów, zlokalizowanych na stanowiskach studenckich, a sterowanych przez dydaktyka z jego mikrokomputera, zlokalizowanego na katedrze. Rozwiązanie takie jest jeszcze droższe i wymaga, oprócz sprzętu, specjalistycznego oprogramowania — daje jednak największe możliwości.

Kadry nauczające informatyki

Rozpowszechnione jest mniemanie, że silną stroną aktualnie realizowanych systemów nauczania informatyki, przy wszystkich sprzętowych i programowych niedogodnościach wymienionych wyżej — są kadry nauczające: kompetentne, ofiarne i fachowe. Podejmując ten temat w niniejszym, z założenia polemicznym artykule, doskonale zdaję sobie sprawę z jego trudności i z kontrowersyjności wszelkich sądów, które zostały tu przytoczone. Nikt bowiem, a już najmniej niżej podpisany, nie może nie zauważyć ofiarności i fachowości nauczycieli zajmujących się nauczaniem informatyki. Jeśli jednak jest tak dobrze, to dlaczego jest tak źle? Wydaje mi się, że za przytoczony na wstępie obraz nieprawidłowego korzystania z techniki obliczeniowej przez „specjalistów hybrydowych" ponoszą odpowiedzialność także nauczający, a może nawet głównie oni.

Sądzę (jakkolwiek jest to oczywiście sąd subiektywny), że nauczyciele (przynajmniej niektórzy) są w niewłaściwy sposób fachowi i zanadto ofiarni. Aby poszerzyć tę tezę, należy zastanowić się, jakie kwalifikacje powodują, że określona osoba cieszy się zasłużonym uznaniem jako wybitny informatyk. Myślę, że nie popełnię dużego błędu, jeśli wskażę trzy grupy tego rodzaju specjalistów.

Pierwsza z nich to teoretycy. Są oni autorami oryginalnych prac z zakresu — na przykład — dowodzenia poprawności programów, organizacji procesów współbieżnych czy teorii translacji. Prace te cieszą się zasłużonym uznaniem i reprezentują bardzo wysoki poziom naukowy, są tak teoretyczne i tak zanurzone w zaawansowanej matematyce, że pomiędzy nimi a codziennym stosowaniem komputera jest przepaść. W istocie, jakkolwiek „nie ma nic bardziej praktycznego niż dobra teoria", to jednak wybitny teoretyk nie musi eo ipso być dobrym praktykiem, a jeszcze może mieć mniej wspólnego z umiejętnością nauczania podstaw praktycznego wykorzystania techniki obliczeniowej.

Drugą grupę tworzą utalentowani programiści. Są to ludzie niezwykli, posiadający talent myślenia w języku komputera i zmuszania go do zadziwiających działań. Jak w wielu innych dziedzinach ludzkiej działalności, tak i w programowaniu komputerów, oprócz wiedzy i rzemiosła konieczna jest pewna wrodzona predyspozycja, powodująca, że program na ten sam temat jeden specjalista napisze szybko i sprawnie, a inny wypracuje mozolnym trudem, przy czym ostateczny efekt w drugim przypadku będzie wielokrotnie gorszy niż w pierwszym. Niewątpliwie pisanie błyskotliwych programów jest sztuką, a ci, którzy ją posiedli, cieszą się zasłużonym szacunkiem i uznaniem środowiska. Czy oznacza to jednak, że stanowią oni optymalny model nauczyciela? Myślę, że z tą opinią można polemizować. Programista, nawet wybitny talent, będzie stawiał programowanie i rozliczne „sztuczki", którymi można w zaskakujący sposób usprawnić proces obliczeń ponad wszelkie inne zajęcia z zakresu techniki obliczeniowej. Będzie rozbudzał ambicje, dla których zaspokojenia nie ma miejsca w codziennej pracy użytkownika środków informatyki; będzie stwarzał sytuację, w której uczeń, przystępując do rozwiązania zadania, będzie w głównej mierze poszukiwał własnego programu, zamiast korzystać z oprogramowania bibliotecznego; przesunie środek ciężkości z problemu „po co komputer stosować" na „jak komputer programować" — problem, który może i powinien być wtórny, mniej ważny.

Trzecia grupa cieszących się poważaniem informatyków jest złożona z projektantów różnego typu systemów informatycznych; obecnie na ogół dość złożonych i „dziwacznych", jako że systemy typowe, dotyczące rutynowych zastosowań, zostały dawno opracowane, zbadane i oprogramowane — wystarczy je kupić lub przystosować. Właśnie ten fakt, że systemy typowe, stanowiące „chleb codzienny" użytkownika komputera, są nisko notowane wśród naukowców, powoduje, że także i w dydaktyce zwraca się większą uwagę na ciekawostki rozwiązań nietypowych i trudnych, poświęcając zbyt mało uwagi systemom typowym i ich stosowaniu. Zapomina się przy tym, że to, co dla specjalisty stanowi elementarz, jest z reguły najbardziej potrzebne początkującym studentom i będzie treścią ich działań po studiach. Tak więc i ta grupa wybitnych informatyków nie zawsze sprawdza się w dydaktyce, gdyż ich własna fachowość zniekształca proporcje potrzebne w pracy nauczycielskiej.

Tak przedstawia się struktura kadry nauczającej informatyki w przypadku ludzi kompetentnych. Dość często jednak nauczyciele są nieprzygotowani...

PODSUMOWANIE

Artykuł eksponuje głównie mankamenty i trudności, celowo pomijając ewidentne sukcesy. Wiadomo, że opracowano szereg mądrych programów i prac metodycznych wskazujących, czego i jak należy uczyć w ramach podstaw informatyki. Na ile jednak te mądre programy są wprowadzane w życie?

Wiadomo, że dostępność sprzętu komputerowego na uczelniach i w szkołach średnich znacznie się poprawiła. Nie ma już uniwersytetu lub politechniki bez własnego komputera, wiele szkół średnich zyskało mikrokomputery, podjęto produkcję sprzętu na potrzeby dydaktyki. Jaka jest jednak proporcja możliwości do potrzeb? Jak jest wykorzystany dostępny sprzęt? Kto i na jakiej podstawie rozdziela komputery?

Również obraz kadry dydaktyków nauczających podstaw informatyki został celowo przedstawiony w krzywym zwierciadle. Problemy przejaskrawiono i celowo pominięto nader liczne przypadki, kiedy wybitny teoretyk czy praktyk potrafi także być doskonałym dydaktykiem. Jak wielu jednak jest takich dobrych dydaktyków? A ilu jest takich, którym brak elementarnych kwalifikacji zawodowych lub którym się zwyczajnie nie chce?

Podobnych pytań jest wiele. Nie na wszystkie łatwo znaleźć poprawną odpowiedź. Jednak stawianie pytań i poszukiwanie poprawnych odpowiedzi (oraz środków zaradczych) jest obowiązkiem każdego, kto widząc obecny stan informatycznej edukacji społeczeństwa rozumie, jakie będą w przyszłości konsekwencje utrzymania tego stanu.

Wielodostępne systemy operacyjne IBM PC/XT (2)

Po omówieniu systemu PICK przedstawiamy dwa kolejne systemy wielodostępne: COHERENT i THEOS.

COHERENT

Wydaje się, że COHERENT jest odmianą siódmej wersji Unixa, później zastąpionej przez System III i System V. Jednak instrukcja obsługi ani słowem nie wspomina o tym, że jest to system operacyjny wzorowany na Unixie. Brak choćby krótkiej wzmianki na temat Unixa jest bardzo niekorzystny, gdyż nie są znane sposoby przenoszenia oprogramowania z systemu COHERENT do Unixa, a byłoby to przydatne w wielu wypadkach.

Jako odmiana Unixa COHERENT jest zadziwiająco kompletny. Zawiera tak wyrafinowane programy, jak yacc — analizator składniowy i awk — język tworzenia raportów, choć brakuje mu wielu poleceń dostępnych w wersji siódmej. Niektóre główne polecenia wersji siódmej zostały usunięte, np. f77 (Fortran), bas (BASIC), **troff** (formater tekstów), **eqn** (program automatycznego składu równań), **tbl** (formater tabel), **lint** (program sprawdzający teksty źródłowe dla kompilatora języka C), **uucp** (program przesyłania plików) i **plot** (program obsługi plotera). Opuszczono także 19 innych poleceń. COHERENT zawiera natomiast 20 poleceń, których nie ma w wersji siódmej Unixa, włączając polecenie **kermit**, które zastępuje **cu** i **uucp** oraz **dos**, a także pozwala odczytywać i zapisywać plik na dyskach elastycznych w systemie MS-DOS (niestety dyski twarde są niedostępne).

COHERENT jest wyposażony w dwa edytory ekranowe **trout** i **elle** bazujące na edytorze EMACS. Różnica pomiędzy nimi nie jest jasna dla autora, gdyż posługiwał się tylko troutem. Jak twierdzą osoby kompetentne, **trout** jest łatwiejszy w użyciu, natomiast **elle** ma większe możliwości. Wydaje się, że oba edytory są znacznie lepiej rozwiązane niż **vi** — edytor Unixa. Przede wszystkim dlatego, że unika się w nich dokuczliwych zmian trybu pracy (**command** i **insert**). Dla osób, które na dobre przywykły do Unixa, a chciałyby niezwłocznie rozpocząć pracę z COHERENTem, dostępny jest także **ed** — edytor wierszowy.

Niektóre polecenia systemu COHERENT mają takie same nazwy jak odpowiedniki Unixa, ale nie są im równoważne. Na przykład polecenie **nroff** (COHERENT) ma znacznie mniejsze możliwości. Spośród 77 opcji w wersji siódmej Unixa **nroff** ma w systemie COHERENT tylko 31 (ale za to najważniejszych).

COHERENT ma wszystkie wywołania systemowe występujące w wersji siódmej Unixa — z wyjątkiem **nice** (ustawiającej priorytet zadania). Wydaje się, że wywołań tych używa się analogicznie w obu systemach. Nie powinno więc nikomu sprawiać kłopotów przenoszenie programów napisanych w języku C między COHERENTem a tą wersją Unixa.

Instalowanie i użytkowanie

Procedura instalowania COHERENTa jest mniej „zautomatyzowana" niż PICKa. Należy podejmować wiele decyzji dotyczących wielkości plików systemowych nie znając skutków, jakie mogą wywołać. Choć ostrzega się, że główna strefa dysku stałego może zostać przepełniona, w instrukcji użytkowania nie ma na ten temat żadnych informacji.

Informacja o pozostawieniu miejsca na dysku na strefę MS-DOS znajduje się na końcu drugiego rozdziału w instrukcji użytkowania; może się więc zdarzyć, że zostanie przeczytana zbyt późno. Najpierw instaluje się COHERENT zostawiając nieco miejsca dla systemu MS-DOS, a następnie używając polecenia FDISK należy utworzyć strefę dyskową dla tego systemu. Jeżeli MS-DOS jest już zainstalowany wcześniej, to trzeba starannie oszacować liczbę cylindrów potrzebną do instalacji COHERENTa. Niestety, systemu COHERENT nie można zainicjować z dysku stałego — trzeba użyć dyskietki.

Wydaje się, że większość osób, które nie są szczegółowo zaznajomione z komputerem PC/XT lub z sytemem DOS, nie będzie w stanie poprawnie zainstalować obok siebie tego systemu i COHERENTa. Jeżeli jednak DOS jest zbędny, to zainstalowanie COHERENTa nie powinno nastręczać jakichkolwiek trudności. System zajmuje cały dysk i sam nadaje odpowiednie wartości parametrom plików systemowych. Instrukcja instalowania nie wyjaśnia, jak połączyć kabel z terminalem, ale wydaje się, że trzy przewody wystarczą (końcówka 1 powinna być połączona bezpośrednio, a 2 skrzyżowana z 3). System może być wyposażony jedynie w terminale VT-52 lub Z-19. Pozostałe mają zbyt mało możliwości na to, aby mogły być zastosowane.

Chociaż COHERENT jest systemem wieloużytkowym, to można nie korzystać z tej możliwości, zwłaszcza podczas uruchamiania programów. Komputer PC/XT nie posiada ochrony pamięci, a tym samym użytkownik bardzo łatwo może doprowadzić do załamania systemu. Jeżeli jednak używa

się programu uruchomieniowego, to nie ma się specjalnie czego obawiać. Należy uważnie kończyć sesję niszcząc wszystkie zadania i wydając polecenie **sync** do dysku, w celu wyzerowania buforów. COHERENT nie ma poleceń zamykania sesji (ang. shutdown), w odróżnieniu od wielu innych implementacji Unixa, ale nic nie stoi na przeszkodzie, aby je napisać.

Można dosyć łatwo odczytywać pliki systemu MS-DOS z dyskietki, jeśli określi się odpowiednie nazwy (identyfikatory) tych plików. W systemie istnieją różne nazwy dla dyskietek jedno- i dwustronnych, a także dla 8- i 9-sektorowych. Te nazwy nie są wyjaśnione w opisie polecenia **dos**, ale w oddzielnym opisie polecenia **fd**.

Jeżeli wystąpią problemy trudne do rozwiązania, a prawdopodobnie to się zdarzy, to można zadzwonić do producenta (na koszt rozmówcy), aby uzyskać pomoc.

Po zainstalowaniu COHERENTa i krótkiej z nim pracy dochodzi się do przekonania, że jest on tak bliski Unixowi, iż należy go zakwalifikować jako jego odmianę. Jeżeli posadzić przy klawiaturze eksperta od Unixa, nie mówiąc mu z czym ma do czynienia, to nie odgadnie, że ma do czynienia z COHERENTem.

Podczas kilku dni testowania COHERENT nie załamał się ani razu, ale były kłopoty w dostaniu się do niego z terminala, który w bardzo krótkim czasie przestawał działać. Wtedy z konsoli systemowej dołączano się na prawach superużytkownika (ang. superuser), niszcząc program zainicjowany z terminala, co powodowało, że terminal wznawiał działanie. Jednak zdarzyło się, że nie można było w ten sposób usunąć powłoki uruchomionej z terminala. Skuteczny był dopiero bezpośredni sygnał zniszczenia programu (ang. kill), a nie programowe zakończenie jego działania. Gdy mamy do czynienia z taką sytuacja, a jesteśmy pewni poprawności działania sprzętu, to wniosek nasuwa się jeden: istnieje jakiś defekt w systemie. Tego samego sprzętu używano przez wiele godzin do testowania PICKa i nie stwierdzono żadnych usterek.

Dokumentacja

Dokumentacja COHERENTa składa się z dwóch opasłych tomów, wydanych w stylu dokumentacji Unixa. Oddzielne strony poświęcono poleceniom, podprogramom, wywołaniom systemowym i plikom specjalnym. Kilka głównych podsystemów ma również swoje własne instrukcje użytkowania (np. **trout** i **nroff**). Niestety, nie ma podręcznika języka C i brak jest informacji o rozmiarach typów, arytmetyce ze znakiem i bez znaku, zmiennych typu rejestrowego i połączeniu z asemblerem. Zamiast tego występuja odsyłacze do oddzielnego podręcznika kompilatora MWC86 (ang. Marc Williams C), który jednakże nie jest dostarczany z COHERENTem.

Efektywność COHERENTa

Wykonano pięć testów w celu sprawdzenia efektywności COHERENTa i porównania go z systemem PC/IX (Unix System III produkcji IBM). Porównano go także z systemami THEOS i MS-DOS (tabela).

W trakcie testowania wystąpiły pewne trudności z programem użytym do czwartego i piątego porównania. Program składał się z 2000 linii w postaci trzech plików napisanych w języku C i realizował algorytm dostępu metodą B-drzewa. Kompilator nie przyjął dwóch z trzech segmentów, mimo iż program ten był wcześniej poprawnie skompilowany przez pięć różnych kompilatorów lub komputerów, począwszy od PC/XT, a skończywszy na VAX-11/780. Kompilator odrzucił pierwszy z plików na skutek fatalnego błędu kompilacji, wysyłając jednocześnie komunikat **no match, op = 65** (brak dopasowania) i drukując ok. tuzina linii informacji dodatkowych, pomocnych przy ponownym uruchamianiu, wyglądających jak drzewo rozbioru gramatycznego. Był to niewątpliwie jeden z najbardziej fascynujących meldunków o błędach, jaki kiedykolwiek udało się widzieć autorowi. Następny plik został odrzucony z powodu błędu **more then 20 stores** (więcej niż 20 bloków pamięci).

Z informacji otrzymanych od serwisu wynika, że pierwszy przypadek świadczy o tym, iż sposób formowania typu (ang. type cast) nie jest przyjmowany przez kompilator, natomiast w drugim wypadku kompilator przekroczył zakres rejestrów. Po wprowadzeniu niewielkich zmian w programie źródłowym wykonano ponowna kompilację i program wynikowy działał poprawnie. Skoro większość użytkowników COHERENTa będzie nastawiona na pisanie programów w języku C, to sytuacja, gdy kompilator nie przyjmuje poprawnie napisanych programów, jest niewątpliwie zniechęcająca. Można się spodziewać pewnych kłopotów przy uruchamianiu takich programów. Ponieważ pracownicy serwisu byli świadomi tej sytuacji, więc kompilator może zostać w przyszłości poprawiony.

Jeśli chodzi o program powłoki (porównanie 1 w tabeli), to zupełnie nie wiadomo dlaczego COHERENT jest znacznie wolniejszy od PC/IX. Natomiast kompilator języka C i konsolidator są znacznie szybsze od swych odpowiedników dla PC/IX oraz Lattice C i MS-DOS, ale skoro kompilatory są różne i na wyjściu dają różne efekty, różnica czasów niekoniecznie mówi o szybkości działania systemów operacyjnych. COHERENT jest prawie tak samo szybki jak PC/IX dla testu B-drzewa oraz bepośredniego we-wy (porównania 2 i 5 w tabeli), ale czasy te są zdeterminowane bardziej pracą dysku niż szybkością działania systemów operacyjnych.

Z porównania wynika, że COHERENT jest wyraźnie wolniejszy od PC/IX, chociaż różnica szybkości wyraźnie zależy od konkretnego zastosowania. Może także okazać się, iż dla wielu uytkowników jest to rzecz zupełnie błaha. Natomiast jeżeli chodzi o szybki kompilator języka C, to dla wielu osób może on być bardzo przydatny.

Wnioski

Jeżeli ktoś chce poznać Unixa, a ma jeszcze nieco wolnego miejsca na swoim PC/XT, to powinien rozważyć możliwość zakupu COHERENTa. Cena 495 dol. — to naprawdę niedużo. Choć COHERENT bazuje na przestarzałej wersji Unixa, różniąc się od niej wieloma szczegółami, to koncepcja systemu — najtrudniejsza rzecz do nauczenia — jest ta sama.

Prawdopodobnie łatwiej jest uruchamiać programy napisane w języku C używajac systemu COHERENT niż MS-DOS, ale jego kompilator wymaga zmian. COHERENT jest słabszy od Unixa ze względu na brak takich narzędzi, jak lint i SCCS (ang. Source Coole Control System — system

Tabela. Porównanie systemów THEOS, COHERENT, PC/IX (Unix System III) i MS-DOS. Czas podano w minutach i sekundach. User oznacza czas procesora przeznaczony na wykonanie instrukcji programu użytkownika, System - czas procesora przeznaczony na obsługę instrukcji programu przez jądro systemu, a Real - upływający czas rzeczywisty (na zegarze ściennym)

Rodzaj testu		THEOS	COHERENT	PC/IX	MS-DOS
Program powłoki (porównanie 1)	User	—	19:24	3:58	—
	System	—	3:25	1:24	—
	Real	—	28:33	9:22	—
We-wy bezpośredniego dostępu (porównanie 2)	User	—	1:44	0:39	—
	System	—	0:20	1:28	—
	Real	2:55	2:35	2:28	3:24
Sekwencyjne we-wy (porównanie 3)	User	—	4:42	1:40	—
	System	—	1:05	0:59	—
	Real	8:57	7:22	3:20	7:07
Kompilacja i konsolidacja programu w języku C (porównanie 4)	User	—	3:52	6:32	—
	System	—	0:32	0:33	—
	Real	—	5:20	8:05	7:18
Zakładanie i instalowanie B-drzewa (porównanie 5)	User	—	0:52	0:43	—
	System	—	0:19	0:20	—
	Real	—	1:43	1:27	2:47

kontroli kodów źródłowych). Jednak kosztuje połowę tego co PC/XT i wymaga mniej pamięci. Mimo że jest mniejszy i działa wolniej, to warto go kupić.

THEOS

Oasis 8 był prawdopodobnie najbardziej wymyślnym systemem operacyjnych dla mikrokomputerów opartych na Z-80. Świetlane lata ma także za sobą znacznie bardziej znany CP/M. THEOS jest rozszerzeniem systemu Oasis 8 dla komputerów opartych na mikroprocesorach 8088 i 8086. Wspomaga on większą liczbę użytkowników, wykonuje więcej zadań, obsługuje większą pamięć RAM i więcej dysków, a także posiada więcej dyrektyw.

Chociaż THEOS ma wiele zalet w porównaniu z systemem MS-DOS, musi równocześnie współzawodniczyć z systemami operacyjnymi przeniesionymi z minikomputerów, takimi jak: PICK, COHERENT i oczywiście Unix. Mówiąc bardziej obrazowo: funkcjonalność THEOSa znajduje się gdzieś między systemami MS-DOS i Unix.

System THEOS w wersji podstawowej składa się z właściwego systemu operacyjnego, tzn. z jądra i współpracujących z nim programów usługowych, tzn. programu obsługi drukarki, programów manipulowania plikami, edytorów tekstowych, programów obsługi poleceń. Języki programowania, pakiety przetwarzania tekstów i bazy danych nie należą do podstawowego oprogramowania THEOSa.

Każdy użytkownik systemu jest przydzielony do stałego obszaru pamięci operacyjnej o wielkości do 64 KB. Obszar danych programu jest ograniczony do rozmiaru tej strefy, lecz jego instrukcje są umieszczane gdzie indziej. Jeżeli dwóch lub więcej użytkowników wykonuje ten sam program, co jest dość powszechne w systemach wielodostępnych, to współdzielą te same instrukcje.

Użytkownik może wykonać program wielozadaniowy składający się z zadania głównego i kilku podzadań. Zadania mogą komunikować się przez zmienne współdzielone i koordynować dostęp do nich przez semafory. Nie ma przesyłania danych ani komunikatów z zadania do zadania, ani potoków.

Wielozadaniowość w systemie THEOS jest obwarowana pewnymi restrykcjami. Pojedynczy użytkownik nie może wykonywać zadania drugoplanowego (ang. background), które nie jest powiązane z zadaniami pierwszoplanowymi (ang. foreground). Nie można, na przykład, skompilować niektórych programów z poziomu drugoplanowego podczas drukowania raportów na poziomie pierwszoplanowym. Aby to wykonać, potrzeba dwóch terminali; wówczas jedna osoba występuje w systemie jako dwóch użytkowników. Następna restrykcja dotyczy niemożności komunikowania się różnych użytkowników. Interpreter poleceń w najmniejszym stopniu nie panuje nad wielozadaniowością, np. przez operator działający na potokach. Zadania współbieżne można konstruować jedynie w BASICu lub w języku C. Wielozadaniowy spooler drukarki wydaje się wyjątkiem, lecz w rzeczywistości tak nie jest, gdyż działa on na takich zasadach jak odrębny użytkownik.

System plików THEOSa składa się z trzech poziomów, ale jest mniej ogólny niż jego odpowiednik dla MS-DOS lub Unixa. Nazwa pliku może składać się z trzech elementów oddzielonych kropkami. Są to: nazwa pliku, typ pliku oraz nazwa biblioteki, do której plik należy. Każdy element może mieć długość ośmiu znaków. Chociaż nie ma pojęcia skorowidza bieżącego (ang. current directory), można tworzyć biblioteki domyślne (ang. default libraries), jak np. link czy macro. Jeżeli tworzy się plik nazwany np. memo, to edytor traktuje go przez domniemanie jako plik biblioteki macro. Gdy ma to być plik samodzielny, to w jego nazwie należy użyć kropki.

Każdy plik posiada właściciela, który może zezwolić na dostęp wszystkim lub tylko sobie. Ze względu na potrzebę użycia hasła w celu zarejestrowania się w sytemie, można go uznać za dość bezpieczny. W systemie istnieją oddzielne zezwolenia na usuwanie, odczyt, zapis i wykonanie plików.

Oprócz sekwencyjnego i bezpośredniego we-wy system plików zapewnia dostęp za pomocą kluczy. Użytkownik może zamknąć dostęp do rekordu, jednakże tylko do jednego w danym czasie i do pliku. Jest to niewłaściwe z punktu widzenia przetwarzania transakcji, gdy aż do momentu jej zakończenia trzeba zamknąć wszystkie rekordy. Niestety, THEOS nie zapewnia innych udogodnień potrzebnych do użytkowania baz danych, jak np. automatyczne przejście do ponownego przebiegu usuniętej (w trybie awaryjnym) transakcji.

Choć możliwe jest zakleszczenie, w podręczniku nie można znaleźć o tym żadnej wzmianki. Nie wykrywa go także system operacyjny. THEOS nie akceptuje plików systemu MS-DOS, choć udostępnia plik systemu Oasis 8. Edytor tekstowy jest kombinacją edytora wierszowego i ekranowego, jego koncepcja podobna jest do zastosowanej w edytorach vi i ex. Przy posługiwaniu się edytorem większość czasu zajmuje praca w reżimie edytora ekranowego, ale takie operacje, jak zmiany w całym programie, wykonuje się w reżimie wierszowym. Dostępne są klawisze specjalne IBM PC/XT (np. ze strzałkami). Nie powinno być kłopotów z nauką i obsługą tego edytora.

Kompilator języka C (nazwany Definitive C) jest opisany w podręczniku jako kompatybilny z systemami Unix III i V. Nie jest to jednak prawdą. Załamuje się on przy użyciu typów enum i void, inaczej działają także funkcje biblioteczne (np. nie ma funkcji fseek). Nie można dołączyć pliku o nazwie stdio.h, ponieważ w bibliotece macro nazwa nie może mieć kropki (należy użyć nazwy stdio). Przyjemne jest, że kompilator zatrzymuje się po wyświetleniu meldunku o błędzie, tak że można go odczytać na przykład po powrocie z kuchni, a następnie przerwać kompilację, jeśli podana informacja jest wystarczająca.

Konsolidator ma również szczególne właściwości. Nazwy modułów wynikowych pojawiają się na ekranie w chwili odnalezienia ich w bibliotece. Choć takie subtelności nie są najważniejsze, to świadczą o jakości i dbałości o szczegóły, dzięki czemu zwiększa się zaufanie do całości systemu. BASIC i VBASIC (z funkcjami graficznymi) są dostępne jako oprogramowanie dodatkowe, przy czym VBASIC współpracuje z wirtualnym sprzężeniem, aby zapewnić obsługę różnorodnych urządzeń wyjściowych.

Instalowanie i użytkowanie

Instalując system THEOS łatwo popełnić błąd. Kierując się instrukcją instalowania można nie osiągnąć zamierzonego celu. Chociaż system zostanie zainstalowany, nadal nie będzie wiadomo, jak go używać. Trzeba zatem poświęcić kilkanaście godzin na przestudiowanie 58-stronicowego podręcznika. Informacje rzeczywiście istotne są w nim tak porozrzucane, że aby uruchomić system, trzeba je starannie wyszukać. Przykładowo, aby uruchomić drukarkę, trzeba znać jej nazwę (centlp). Choć podręcznik nie mówi o tym wprost, można z niego dowiedzieć się, gdzie znaleźć plik z nazwami urządzeń fizycznych. Należy też wiedzieć, jak działa polecenie ATTACH. Choć istnieje rozdział pt. „Dołączanie urządzeń drukujących", to jest on poświęcony omówieniu zasad ogólnych i odsyła czytelnika do rozdziału „Użytkowanie spoolera drukarki", którego nie ma w podręczniku (!), ale jest inny, zatytułowany „Użycie drukarki". Po zmarnowaniu jeszcze jakiegoś czasu czytelnik zdaje sobie sprawę, że znacznie łatwiej byłoby przeczytać kolejno wszystkie rozdziały, starając się jednocześnie jak najwięcej zapamiętać. Po takiej lekturze przynajmniej wiadomo, gdzie szukać żądanych informacji.

System THEOS można sprowadzić z dysku stałego, ale bardzo trudno odczytać z instrukcji, jak to wykonać. Należy użyć dyskowego programu usługowego z opcja write boot sectors (zapisz sektory inicjujące). Wiele kłopotów sprawia także odczytanie, jak zainstalować drugi terminal, ale gdy uda się to zrozumieć, samo instalowanie przebiega dość gładko. Eksperci od fizycznych i logicznych nazw urządzeń, kodów itp., powinni instalować system THEOS błyskawicznie. Mimo iż nie ma diagramu połączenia kabli, do pracy wystarcza taka sama konfiguracja urządzeń jak dla COHERENTa. Jeżeli sy-

stem zostanie zainstalowany i uruchomiony przez eksperta, to można rzec, iż jest łatwiejszy do użycia od COHERENTa lub Unixa (zwłaszcza że omija się kłopoty związane z dokumentacją). Jest to jednak złudzenie, gdyż THEOS posiada wiele poleceń, z których każde ma pewną liczbę opcji, a ponadto sama konwersja z nim jest nieco zawiła. Reasumując, THEOS nie jest tak zgrabnym systemem jak Unix.

Dokumentacja

Dokumentacja jest napisana językiem jasnym i prostym, ale organizacja informacji i posługiwania się nią nastręcza sporo kłopotów. Przede wszystkim brakuje jej nieco rozwlekłości stylu. Na przykład zamiast podać kody terminala w punkcie „Console Class Codes" należącym do rozdziału „Attaching Console Terminals", podręcznik odsyła do rozdziału „Console Terminal Class Codes", który nie istnieje. W ten sposób można narazić się na daremne poszukiwania podążając za odsyłaczami.

Jednocześnie w wielu miejscach podręcznik jest zbyt rozwlekły. Długie rozprawy na temat filozofii wielozadaniowości są pomieszane z naprawdę istotnymi informacjami. Ponieważ odsyłacze często prowadzą donikąd, a w dodatku brakuje indeksów, należy czytać wszystko po kolei. Zajmuje to dwukrotnie więcej czasu, gdyż należy brnąć przez sterty informacji, które można by spokojnie przeczytać w wolnej chwili.

Podręcznik jest adresowany do niewłaściwych odbiorców. Jeden z rozdziałów zajmuje się dyskusją nad opóźnieniem ładowania głowic dysku stałego, a następny tłumaczy, co należy rozumieć przez słowo „abbervation", uzupełniając to wyjaśnienie przykładem. Co więcej, temat opóźnienia ładowania głowic ma pierwszeństwo przed opisem instalowania drukarki.

Podsumowując, podręcznik nie prezentuje logicznie uporządkowanej informacji, ma zbyt wiele odsyłaczy (w wielu wypadkach błędnych), miesza informacje istotne z nieistotnymi, nie posiada indeksów, ale po przeczytaniu całości można zrozumieć, jak działa THEOS, choć mogą być kłopoty z zainstalowaniem i używaniem systemu.

Funkcjonowanie THEOSa

W systemie THEOS nie działał program B-drzewa (porównanie 5 z tabeli), ponieważ THEOS nie odczytuje programów źródłowych zapisanych w systemie MS-DOS (należy je przekodować). Zamiast tego, sprawdzono działanie dostępu bezpośredniego i sekwencyjnego. Na podstawie testów wydaje się, że we-wy bezpośredniego dostępu jest szybsze niż w systemie MS-DOS i nieco tylko wolniejsze od COHERENTa i PC/IX. We-wy sekwencyjne jest znacznie wolniejsze od tych systemów, z którymi dokonano porównania (patrz tabela).

Trudno jest ekstrapolować wnioski na podstawie jedynie dwóch programów, ale bezpiecznie będzie powiedzieć, że system plików THEOSa nie jest szybki i można go sklasyfikować razem z bazą danych systemu MS-DOS. Ze względu na możliwość dostępu indeksowego do rekordów, THEOS może być szybki, ale nie przeprowadzono żadnego testu w celu sprawdzenia tego przypuszczenia.

WNIOSKI

THEOS jest systemem dobrze zaprojektowanym i dobrze wykonującym swoje zadania. Łatwiej jest go zrozumieć i używać niż Unixa, ma jednak mniej możliwości. Wdrażając konkretne zastosowanie użytkownik nie interesuje się liczbą możliwości, lecz dostępnością tych, których potrzebuje. THEOS może okazać się wspaniałym narzędziem dostosowanym do wielu potrzeb.

W porównaniu z innymi systemami operacyjnymi zrobionymi dla PC/XT cena THEOSa wydaje się zbyt wysoka, jako że kosztuje on dwa razy tyle co PICK lub COHERENT, a razem z kompilatorem języka C — więcej niż PC/IX.

Oprac. PIOTR STRUTYŃSKI
na podst. BYTE, No. 9, 1985

Targi Lipskie '86

Ekspozycja informatyczna na Wiosennych Targach Lipskich (LFM) jest tradycyjną okazją do kompleksowej prezentacji oferty gospodarzy, a właściwie Kombinatu Robotron. Oferta ta stała się już tak wszechstronna i nadążająca za potrzebami rynku wewnętrznego, że powoduje nie tylko zanik zainteresowania producentów zachodnich, ale w pewnym stopniu również wystawców z innych krajów RWPG. Klasycznym przykładem tego zjawiska jest malejący udział w tej imprezie polskiego przemysłu komputerowego. Dlatego nieco spóźnioną relację z LFM '86 należy traktować jako okazję do poznania aktualnej treści oferty NRD.

W najbardziej nas interesującej grupie sprzętu mikrokomputerowego na czoło wysuwa się 16-bitowy komputer profesjonalny A 7100, wyposażony w pamięć wewnętrzną o pojemności do 768 KB oraz dwie wbudowane stacje dyskietek 5 1/4 cala (po 500 KB). Rozszerzalność systemu zapewnia realizację różnych zastosowań, m.in. wymagających użycia grafiki o dużej rozdzielczości (640×400). Podstawą oprogramowania jest kompatybilny z CP/M system operacyjny SCP 1700 oraz znormalizowane wersje języków Basic i Fortran.

W kategorii komputerów 8-bitowych oferowany jest PC 1700 z pamięcią 64 KB i możliwością dołączania urządzeń zewnętrznych, w tym różnych typów drukarek wytwarzanych przez tego samego producenta (Zakłady Sömmerda). Systemami operacyjnymi są wspomniany już SCP 1700 oraz BROS, stosowany dotąd w komputerze biurowym A 5110. Przekonywającym dowodem szerokich możliwości zastosowań PC 1700 była demonstracja działania systemu relacyjnej bazy danych REDABAS.

W kategorii komputerów biurowych zaprezentowano nowy model A 5120.16, wyposażony w procesor 16-bitowy oraz pamięć 250 KB. Zastosowanie koprocesora arytmetycznego pozwala efektywnie wykonywać na tym sprzęcie również złożone obliczenia. Bardzo obszerne oprogramowanie obejmuje wszystkie najważniejsze języki programowania (w tym Pascal oraz C) oraz liczne pakiety użytkowe (przetwarzanie tekstów, obliczania tabelaryczne, systemy bazy danych). Podkreśla się, że rozwiązania sprzętowe i programowe wszystkich omówionych modeli pozwalają na ich stosowanie jako terminali sprzętu Jednolitego Systemu.

Ofertę w tej grupie zamyka komputer domowy KC 85/1, proponowany głównie do zastosowań dydaktycznych. Może on współpracować z telewizorem, magnetofonem kasetowym, elektryczną maszyną do pisania Erika S 6005 oraz drukarką termiczną Robotron K 6303.

Silnym akcentem ekspozycji, a jednocześnie dowodem nadążania przez NRD za światowymi trendami, była prezentacja działania sieci lokalnej BÜRONET 1, opracowanej zgodnie z najnowszymi zaleceniami ISO.

Szczególnie szeroki wachlarz rozwiązań prezentowano w dziedzinie zastosowań CAD/CAM: od wyspecjalizowanych terminali, współpracujących ze sprzętem Jednolitego Systemu, do indywidualnych, konwersacyjnych stanowisk pracy, opartych na sprzęcie mini- i mikrokomputerowym. Oprócz już znanych, ale znacznie zmodernizowanych konstrukcji pokazano po raz pierwszy inteligentny terminal graficzny K 8918, digitajzer K 6404, ploter K 6411 (format A2) oraz system rejestracji danych produkcyjnych A 5222, umożliwiający wprowadzenie również informacji graficznych.

Całkowicie nowym i nie spotykanym w tej skali punktem ekspozycji NRD była imponująca oferta produktów programowych, jaką prezentowało wydzielone w ostatnim czasie z Kombinatu przedsiębiorstwo Robotron-Projekt. Oferta obejmowała pełną skalę oprogramowania podstawowego, narzędziowego i użytkowego dla wszystkich rodzajów produkowanego sprzętu. W materiałach informacyjnych podkreślono, że oprogramowanie uwzględnia aktualne międzynarodowe zalecenia normalizacyjne oraz obejmuje programy użytkowe, zaliczane do obowiązkowego wyposażenia 16-bitowych komputerów osobistych.

Terminologia lokalnych sieci komputerowych (3)

Przyglądając się dokładniej słownictwu używanemu w dziedzinie lokalnych sieci komputerowych, którego omawianie rozpoczęto w numerze 11—12/86 »Informatyki«, nietrudno spostrzec ogromne bogactwo nowych terminów i pojęć; mnogość ich wystarczyłaby do zapełnienia kilku kolejnych rubryk terminologicznych. Jest ich tak dużo, że sprawiają wrażenie istnienia pewnego chaosu terminologicznego, szczególnie dla osób nie zajmujących się zawodowo tą dziedziną informatyki. Należy przyjąć pewną systematykę w przedstawianiu tych dość złożonych kwestii terminologicznych. Z tego względu omawianie tej problematyki rozpoczęto od modelu odniesienia ISO, który wszakże jest modelem tylko na poziomie logicznym. W drugim odcinku przedstawiono te zagadnienia od strony fizycznej, tj. przestrzeni (konfiguracje, ośrodki transmisji) i czasu (rodzaje transmisji i metody dostępu do ośrodka).

Stanowi to wystarczające ramy do wprowadzenia następnej partii terminów, odnoszących się wreszcie do istoty sieci komputerowych, tj. przesyłania danych (wiadomości). Podstawową rolę w przesyłaniu wiadomości odgrywają protokoły transmisji.

Takim protokołem transmisji na najniższym poziomie jest fizyczny sposób kodowania informacji dwójkowej, tj. poszczególnych bitów. Choć istnieje wiele różnych możliwości fizycznego przesyłania informacji dwójkowej, w sieciach lokalnych upowszechniło się tzw. kodowanie w systemie Manchester (ang. Manchester encoding), zwane w telekomunikacji kodowaniem bifazowym. Według definicji podanej w jednej z wcześniejszych propozycji normy IEEE 802, kodowaniem bifazowym nazywa się metodę łącznego kodowania sygnału zegarowego i danych na pola bitowe (ang. bit cell), polegającą na podziale każdego pola bitowego na połowy o przeciwnej polaryzacji; zero logiczne jest reprezentowane przez wysoki poziom sygnału w pierwszej połowie pola bitowego i niski poziom sygnału w drugiej połowie, a jedynka logiczna — przez niski poziom sygnału w pierwszej połowie pola i wysoki — w drugiej. W normie IEEE 802.3-1985 sygnał CD0, ang. clocked data zero (i odpowiednio, sygnał CD1, ang. clocked data one), reprezentują zero logiczne (jedynkę logiczną) zakodowane bifazowo jako poziom wysoki w pierwszej połowie pola bitowego i poziom niski w drugiej połowie pola (odpowiednio, poziom niski w pierwsze połowie pola bitowego i poziom wysoki w drugiej połowie pola).

Oprócz zwykłego kodowania bifazowego często stosuje się tzw. różnicowe kodowanie bifazowe (ang. differential Manchester encoding), w którym każde pole bitowe dzieli się na połowy o przeciwnej polaryzacji, jednak zero logiczne jest reprezentowane przez zmianę polaryzacji na początku pola, a jedynka logiczna przez brak zmiany polaryzacji na początku pola.

Podobnie jak bit na poziomie fizycznym, elementarną jednostką transmisji na poziomie łącza jest tzw. ramka (ang. frame), tj. uporządkowany ciąg bitów charakterystyczny dla określonego protokołu transmisji. Ramka jest jednym z ważniejszych pojęć we wszelkich sieciach komputerowych. Ramka sieci IEEE 802.3 składa się z następujących pól o długościach będących całkowitymi wielokrotnościami bajtów (w normach określanych jako oktety, ang. octets):

- **preambuły** (ang. preamble), tj. siedmiobajtowego pola umożliwiającego układom PLS (p. część 1 w 11—12/86 numerze »Informatyki«) warstwy fizycznej zsynchronizowanie taktowania z odbieraną ramką,
- **ogranicznika początkowego ramki** (ang. start frame delimiter, SFD), 10101011, w literaturze polskiej zwanego niekiedy bajtem flagowym,
- **adresu docelowego** (2 lub 6 bajtów), który oprócz adresu przeznaczenia zawiera bit określający, czy jest to adres indywidualny czy grupowy (tzn. uwzględniający rozgłaszanie częściowe lub pełne) oraz bit określający czy jest to adres globalny czy lokalny,
- **adresu źródłowego** o formacie podobnym do adresu docelowego,
- **długości pola danych** (2 bajty),
- **pola danych** (od 40 do 1500 bajtów),
- **wypełnienia** (ang. pad), dodawanego w wypadku, gdy rzeczywista długość danych jest mniejsza od minimalnej wymaganej do poprawnej pracy protokołu transmisji,
- **ciągu kontrolnego** (ang. frame check sequence, FCS), służącego do kontroli poprawności transmisji metodą CRC (ang. cyclic redundancy check); jego wartość jest funkcją zawartości wszystkich pól ramki oprócz preambuły, SFD i FCS.

W adresie docelowym ramki uwzględnia się fakt, że oprócz transmisji dwupunktowej (ang. point-to-point) może wystąpić tzw. rozgłaszanie (częściowe lub pełne). Rozgłaszanie (ang. broadcast) polega na transmitowaniu jednostki danych (na poziomie fizycznym jest to pojedynczy sygnał, a na poziomie łącza — pojedyncza ramka) do wszystkich stacji dołączonych do ośrodka transmisji sieci lokalnej. Rozgłaszanie częściowe (ang. multicast) polega na transmitowaniu pojedynczej jednostki danych do określonego podzbioru stacji sieci lokalnej.

Przy transmisji na poziomie fizycznym lub łącza występuje wiele zjawisk nazwanych już po angielsku, lecz nie zawsze mających polskie odpowiedniki. Przykładowo, wśród wielu rodzajów zniekształceń sygnału możliwe jest rozsynchronizowanie (ang. jitter, timing jitter) polegające na utracie przez sygnał informacyjny właściwego położenia względem sygnału odniesienia (sygnału taktowania), co może prowadzić do błędów, szczególnie przy dużej szybkości transmisji. Aby zapobiec pewnym rodzajom zniekształceń przy przesyłaniu kolejnych ramek, stosuje się tzw. odstęp międzyramkowy (ang. interframe gap), tj. wymuszony okres bezczynności (ang. idle time) między transmisją kolejnych ramek, utrzymywany w celu pozwolenia na ustalenie się procesom zachodzącym w sterownikach i w kanale fizycznym.

Do niekorzystnych zjawisk występujących w sieciach lokalnych należy też tzw. rozwlekanie (ang. jabber), tj. błąd polegający na ciągłym nadawaniu danych przez stację ponad ustalony limit. Innym szkodliwym zjawiskiem jest tzw. przeciążenie (ang. congestion), tj. graniczny stan sieci określony przez warunki, w których natężenie ruchu (ang. traffic) w sieci jest ograniczone przez jej możliwości przepustowe.

Charakterystyczną cechą niektórych rodzajów sieci jest tzw. rywalizacja (ang. contention), tj. metoda dostępu polegająca na współzawodnictwie dwóch lub kilku stacji o współdzielony ośrodek transmisji. Warto przypomnieć, że przez metodę dostępu (ang. access method) w sieciach komputerowych rozumie się mechanizm podziału wspólnego fizycznego ośrodka transmisji między wielu użytkowników. Przy rywalizacyjnej metodzie dostępu możliwe są kolizje, których znaczenie omówiono w poprzednim odcinku tego cyklu. Jednym ze sposobów zwalczania kolizji jest tzw. zagłuszanie (ang. jamming), polegające na celowym przesyłaniu zakodowanego ciągu bitów, służącego do wzmocnienia kolizji, w wypadku jej wystąpienia, w celu zapewnienia wykrycia i odrzucenia przez wszystkie stacje.

Każda warstwa modelu odniesienia sieci, scharakteryzowanego wstępnie w pierwszym odcinku tego cyklu, odgrywa określoną rolę wobec pozostałych warstw. Rola warstwy jest opisana przez jej funkcje i usługi komunikacyjne. O ile funkcje komunikacyjne mają charakter głównie we-

wnętrzny dla danej warstwy, to usługi występują na poziomie sprzężenia warstw i stanowią zadania spełniane wobec warstwy wyższej. Przykładowo, głównym zadaniem warstwy fizycznej jest sterowanie przepływem bitów, a do usług zapewnianych przez warstwę łącza należy sterowanie dostępem do ośrodka transmisji oraz detekcja i korekcja błędów.

Na poziomie łącza danych wyróżnia się usługi połączeniowe i bezpołączeniowe. Usługa bezpołączeniowa (ang. connectionless service) jest to sposób, w jaki obiekty sieci mogą wymieniać jednostki danych bez ustanowienia połączenia na poziomie łącza danych. Usługa połączeniowa może być dwupunktowa (ang. point-to-point), częściowo rozgłaszana (ang. multicast) lub w pełni rozgłaszana (ang. broadcast). Usługi połączeniowe (ang. connection-oriented services) stanowią zbiór usług służących obiektom sieci do ustanowienia, używania i zakończenia połączeń na poziomie łącza danych. Dla usług połączeniowych określa się jedynie połączenia punktowe.

Za jednostkę danych na poziomie warstwy sieciowej uważa się pakiet (ang. packet). Niestety, nie udało mi się znaleźć w miarę precyzyjnej definicji pakietu, dlatego podaję jedynie określenie opisowe. Przez pakiet rozumie się blok danych występujący w ściśle zdefiniowanej postaci zawierającej nagłówek. Maksymalna wielkość pakietu jest ustalona, a dłuższe wiadomości przesyła się przez sieć w postaci kilku pakietów.

Do usług zapewnianych przez warstwę sieciową należy m. in. multipleksowanie, kontrola błędów, a także sterowanie przepływem danych i trasowanie. Sterowanie przepływem danych (ang. data flow control) rozumiane jako dostosowanie tempa przesyłania ramek do możliwości odbiorczych stacji w warstwie sieciowej, bywa też zaliczane do usług warstwy łącza.

Trasowaniem (ang. routing) nazywa się przydział trasy telekomunikacyjnej, po której wiadomość lub wywołanie dociera do miejsca przeznaczenia. Zazwyczaj jest to funkcja odpowiedniej warstwy modelu odniesienia ISO, polegająca na przetłumaczeniu nazwy lub adresu miejsca przeznaczenia (stacji) na drogę, po której ma ono być osiągnięte. Trasa (ang. route) jest to droga między stacją nadawczą a odbiorczą wybrana w celu przesyłania informacji. Z różnych rodzajów trasowania warto wymienić trasowanie skorowidzowe i rozpływowe. Trasowanie skorowidzowe, zwane też tablicowym (ang. directory routing), jest to metoda wyboru drogi wiadomości lub pakietu, polegająca na użyciu w każdym węźle skorowidza adresowego, określającego dla każdego miejsca przenaczenia zalecany kierunek wyjściowy. Skorowidz może również wskazywać drogi alternatywne. Trasowanie rozpływowe (ang. flooding) jest metodą wyboru trasy przesyłania pakietów, polegającą na powielaniu pakietu i wysyłaniu go do wszystkich węzłów, zapewniając w ten sposób osiągnięcie pożądanego miejsca przeznaczenia.

Z różnych rodzajów usług warto jeszcze wymienić tzw. usługi datagramowe (ang. datagram service), tj. metodę przesyłania wiadomości zawartej w polu informacyjnym pakietu do miejsca przeznaczenia zdefiniowanego przez jego pole adresowe. Do najważniejszych, bo globalnych, zadań realizowanych w sieci, należy tzw. zarządzanie siecią (ang. network management). Zarządzanie siecią jest to funkcja stacji, odnosząca się do wszystkich warstw protokołów komunikacyjnych, polegająca na ustawianiu parametrów kontrolnych, uzyskiwaniu raportów o błędach i decydowaniu o dołączeniu stacji lub odłączeniu jej od ośrodka.

JANUSZ ZALEWSKI

WARUNKI PRENUMERATY NA 1987 R.

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

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.
Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

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

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

Przedpłaty na prenumeratę przyjmowane są w terminach:
— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

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

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

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

Cena prenumeraty INFORMATYKI (w złotych)					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
450,—	150,—	900,—	300,—	1800,—	600,—

Lukaszewicz W.: Przechowywanie wiedzy i niewiedzy w bazach danych (1)

INFORMATYKA 1987, Nr 2, s. 1

Charakterystyka podejścia logicznego przy tworzeniu baz wiedzy. Omówiono logiczną reprezentację wiedzy oraz problemy wnioskowania niemonotonicznego.

Лукашевич В.: Хранение знаний и незнания в базах данных (1)

INFORMATYKA 1987, № 2, с. 1

Характеристика логического подхода при создании баз знаний. Описание логического представления знаний и проблемы немонотонического предложения.

Tabak D.: Intel 80386 i nowe mikroprocesory 32-bitowe (2)

INFORMATYKA 1987, Nr 2, s. 4

Druga część charakterystyki mikroprocesorów 32-bitowych, zawierająca organizację mikroprocesora 80386 i jego porównanie z innymi mikroprocesorami 32-bitowymi.

Табак Д.: INTEL 80386 и новые 32-разрядные микропроцессоры (2)

INFORMATYKA 1987, № 2, с. 4

Вторая часть характеристики 32-разрядных микропроцессоров, содержащая организацию микропроцессора 80386 и его сравнение с другими 32-разрядными микропроцессорами.

Owczarczyk J., Stolarski M., Woźniak E.: Procesory tablicowe — współbieżne systemy przetwarzania obrazów

INFORMATYKA 1987, Nr 2, s. 8

Charakterystyka procesorów tablicowych zawierająca omówienie ich architektury, dotychczasowych realizacji oraz najnowszych tendencji rozwojowych.

Овчарчик Я., Столярский М., Возняк Э.: Стендовые процессоры — параллельные системы обработки изображений

INFORMATYKA 1987, № 2, с. 8

Характеристика стендовых процессоров, содержащая описание их архитектуры, существующих конструкций и новейших тенденций развития.

Skolimowski J.: Specjalizowany system mikrokomputerowy SIGMA-3

INFORMATYKA 1987, Nr 2, s. 12

Charakterystyka systemu umożliwiającego tworzenie mikrokomputerów do zastosowań profesjonalnych. Omówiono konstrukcję sprzętu i oprogramowanie oraz wykazano różnice pomiędzy uniwersalnymi mikrokomputerami bez pamięci dyskowej.

Сколимовский Я.: Специализированная система микро ЭВМ SIGMA-3

INFORMATYKA 1987, № 2, с. 12

Характеристика системы, позволяющей создавать микро ЭВМ для профессиональных областей применения. Описание конструкции машин и программного обеспечения; различия между универсальными микро ЭВМ без дисковой памяти.

Gondzio M.: Sieci Petriego (2). Przekształcenia i rodzaje sieci

INFORMATYKA 1987, Nr 2, s. 13

Druga część charakterystyki sieci Petriego, zawierająca omówienie ich modyfikacji oraz typowych zastosowań.

Гондзио М.: Сети Петри (2). Преобразования и виды сетей

INFORMATYKA 1987, № 2, с. 13

Вторая часть характеристики сетей Петри, содержащая описание их модификаций и типовых областей применения.

Grabowicz R., Zalewski J.: System operacyjny PC-DOS (5)

INFORMATYKA 1987, Nr 2, s. 17

Zakończenie charakterystyki systemu PC-DOS, zawierające omówienie oprogramowania służącego do przygotowania programów użytkowych (edytor, makroasembler, konsolidator, program uruchomieniowy).

Грабович Р., Залевский Я.: Операционная система PC-DOS (5)

INFORMATYKA 1987, № 2, с. 17

Завершение характеристики системы PC-DOS, содержащая описание программного обеспечения, предназначенного для разработки прикладных программ (эдитор, макроассемблер, консолидатор, программа запуска).

Pawłowski M.: Programator pamięci UVEPROM serii 27XXX

INFORMATYKA 1987, Nr 2, s. 20

Charakterystyka programatora pamięci UVEPROM opracowanego w Instytucie Informatyki Politechniki Warszawskiej.

Павловский М.: Программатор памяти UVEPROM серии 27XXX

INFORMATYKA 1987, № 2, с. 20

Характеристика программатора памяти UVEPROM, разработанного Институтом информатики Варшавского политехнического института.

Tadeusiewicz R.: Nauczanie informatyki

INFORMATYKA 1987, Nr 2, s. 22

Krytyczna ocena aktualnego stanu nauczania informatyki w szkołach średnich i wyższych oraz propozycje poprawy sytuacji.

Тадеусевич Р.: Обучение информатике

INFORMATYKA 1987, № 2, с. 22

Критическая оценка настоящего состояния в области обучения информатике в средних школах и вузах, а также предложения по улучшению существующего положения.

Łukaszewicz W.: Knowledge and ignorance storaging in data bases (1)

INFORMATYKA 1987, No. 2, p. 1

Characteristics of logical approach to building knowledge bases. Logical knowledge representation and nonmonotonic concluding problem are discussed.

Tabak D.: Intel 80386 and new 32-bit microprocessors (2)

INFORMATYKA 1987, No. 2, p. 4

Second part of 32-bit microprocessor characteristics, which includes discussion of the Intel 80386 organization and its comparison with other 32-bit microprocessors.

Owczarczyk J., Stolarski M., Woźniak E.: Array processors — concurrent image processing systems

INFORMATYKA 1987, No. 2, p. 8

Characteristics of array processors, which includes discussion of their architecture, existing realizations and newest development trends.

Skolimowski J.: SIGMA-3 — a specialized microcomputer system

INFORMATYKA 1987, No. 2, p. 12

Characteristics of the system, which enables building microprocessor for professional applications. Hardware and software, as well as difference between universal microcomputers without disc storage, are discussed.

Gondzio M.: Petri nets (2). Modification and net types

INFORMATYKA 1987, No. 2, p. 13

Second part of Petri net characteristics, which includes discussion of selected modifications and typical applications.

Grabowicz R., Zalewski J.: PC-DOS operating system (5)

INFORMATYKA 1987, No. 2, p. 17

Termination of the PC-DOS characteristics, which includes discussion of software for application programs preparation (editor, macroassembler, consolidator, debugger).

Pawłowski M.: Programmer for 27XXX series UVEPROM store

INFORMATYKA 1987, No. 2, p. 20

Characteristics of the UVEPROM store programmer, developed in Informatics of Warsaw Institute Technology.

Tadeusiewicz R.: Informatics teaching

INFORMATYKA 1987, No. 2, p. 22

Critical evaluation of actual situation of informatics teaching in middle and high schools, as well as improvement proposals.

Łukaszewicz W.: Wissenschafts- und Ignoranzspeicherung in Datenbanken (1)

INFORMATYKA 1987, Nr. 2, S. 1

Eine Charakteristik der logischen Auffassung zur Bildung der Wissenschaftsbanken. Es wurde die logische Wissenschaftsrepräsentation und die Probleme der nichtmonothonischen Beantragung besprochen.

Tabak D.: Intel 80386 und neue 32-bit-Mikroprozessoren (2)

INFORMATYKA 1987, Nr. 2, S. 4

Zweiter Teil einer Charakteristik von 32-bit-Mikroprozessoren, der die Intel 80386-Organisation und sein Vergleich mit anderen 32-bit-Mikroprozessoren umfast.

Owczarczyk J., Stolarski M., Woźniak E.: Tafelprozessoren — simultane Bildverarbeitungssysteme

INFORMATYKA 1987, Nr. 2, S. 8

Eine Charakteristik von Tafelprozessoren, die eine Besprechung ihrer Architektur, der bisherigen Realisationen und der neuesten Entwicklungstendenzen, umfasst.

Skolimowski J.: SIGMA-3 — ein spezialisiertes Mikrocomputersystem

INFORMATYKA 1987, Nr. 2, S. 12

Eine Charakteristik des Systems für Mikrocomputerbildung im Bereich der professionellen Anwendungen. Es wurde Hard- und Software, sowie Unterschiede zwischen universellen Mikrocomputern ohne Plattenspeicher, besprochen.

Gondzio M.: Petri Netze (2). Modifikationen und Netztypen

INFORMATYKA 1987, Nr. 2, S. 13

Zweiter Teil einer Charakteristik von Petri Netzen, der eine Besprechung von ausgewählten Modifikationen und typischen Anwendungen, umfasst.

Grabowicz R., Zalewski J.: PC-DOS-Betriebssystem (5)

INFORMATYKA 1987, Nr. 2, S. 17

Beendigung einer Charakteristik von PC-DOS. Es wurde die Software für Vorbereitung der Anwendungsprogrammen (Editor, Mikroassembler, Konsolidator, Debugger) besprochen.

Pawłowski M.: Programmator für UVEPROM-Speicher der 27XXX-Serie

INFORMATYKA 1987, Nr. 2, S. 20

Eine Charakteristik des Programmators für UVEPROM-Speicher, der im Institut für Informatik der Warschauer Technischen Universität erarbeitet wurde.

Tadeusiewicz R.: Informatik-Unterricht

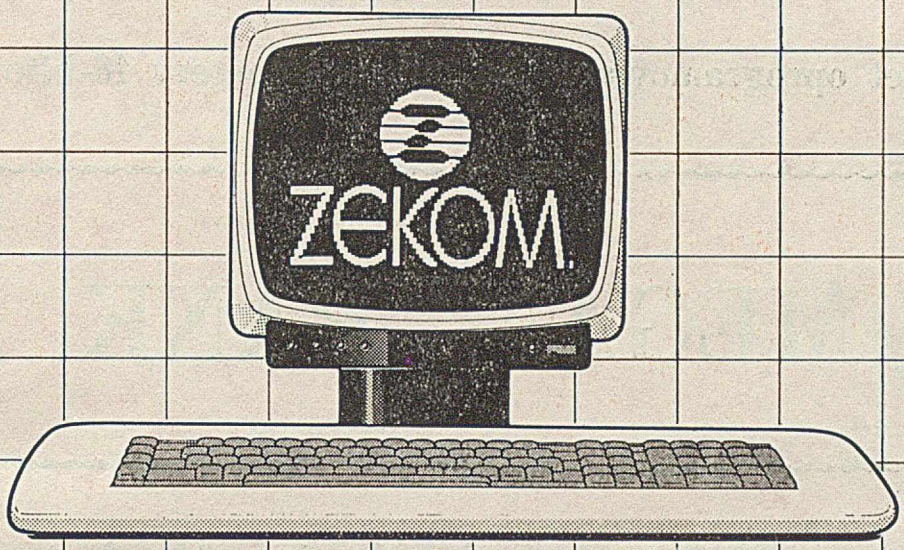
INFORMATYKA 1987, Nr. 2, S. 22

Eine kritische Beurteilung der heutigen Lage von Informatik-Unterricht in Ober- und Hochschulen, sowie Vorschläge zur Besserung der Situation.

Jeśli interesuje Państwa

* Profesjonalny sprzęt o wysokiej jakości, niezawodny w eksploatacji
* sprawny serwis
* krótkie terminy dostaw, lub dostawy natychmiastowe

TO WYROBY ZEKOMU SĄ DO PAŃSTWA DYSPOZYCJI



Zakład Elektroniki Komputerowej

oferuje:

Terminale nadawczo-odbiorcze

MV 2580 Standard VT 52 firmy DEC/ odpowiedniki MERA 7953. Przeznaczone do pracy w systemach komputerowych wyposażonych w kanał transmisji V 24 lub pętli prądowej 20/60 mA – jako końcówki zdalnego dostępu.

Terminale nadawczo-odbiorcze

MV 2581 Odpowiedniki MERA 7911 N. Przeznaczone do pracy w systemach komputerowych ODRA 1300 wyposażonych w jednostkę sterującą MERA 7802.

Monitory ekranowe

MV 2580L Przeznaczone do współpracy z drukarką DZM-180 oraz jej wersjami rozwojowymi typu DZM – 180/KSR. Zaoszczędzają czas operatora, papier i zwiększają dyspozycyjność drukarki (konsoli).

Sprzęgi

SV 24/400 Odpowiedniki UZDAT-11 systemu komputerowego MERA 400. Umożliwiają obsługę przerwań operatorskich.

ZEKOM ZAKŁAD ELEKTRONIKI KOMPUTEROWEJ
ul. Makowa 8, 91-480 Łódź tel. 34 30 49

EO/1391/K/86

KONIEC TWOICH PROBLEMÓW

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

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed
Twoim przedsiębiorstwem.

Wyobraź sobie

MILIARD

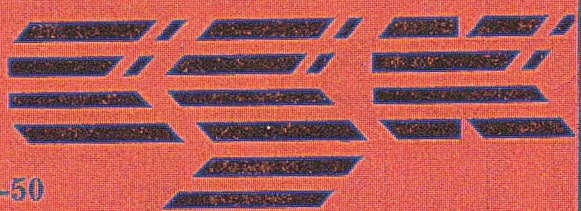
rekordów, które możesz
zapełnić według własnych potrzeb i uznania.

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

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!

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



EO/74/KJ87