mikroKLAN 15

3 1985

informatyka

Funkcje oprogramowania zarządzającego

Przegląd mechanizmów komunikacji

Kompilator LOGLANU 82 dla MERY 400

WYDAWNICTWO NOT SIGMA

Cena 100 zł

informatyka

3 1985

Nr 3

Miesięcznik Rok XX

Marzec 1985

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

Materiałów nie zamówionych redakcja nie zwraca

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

Zakł. Graf. „Tamka". Zam. 1341-1300/84. Obj. 4,0 ark. druk. Nakład 6500 egz. N-13.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza zł 100,—
Prenumerata roczna zł 1200,—

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Julian Winiewski o języku C
- Frank Land o systemach informacyjnych
- Wojciech Cellary i Jan Węglarz o nauczaniu mikroinformatyki
- Wacław Iszkowski o superkomputerze CRAY
- Jan Bielecki o standardzie operacji wejścia-wyjścia w ADZIE
- Jerzy Karczmarczuk o efektywności obliczeń numerycznych na komputerach SINCLAIRA
- Halina Kontkiewicz-Chachulska o SICOBIE '84
- Jacek Irlik o kwalifikacji prawnej programu komputerowego

ZBIGNIEW KIERZKOWSKI
JACEK MAŁUSZYŃSKI
Poznań

Zarządzanie współbieżną aktualizacją bazy danych (1)

Funkcje oprogramowania zarządzającego

Zwiększenie efektywności działania systemów informatycznych można uzyskać przez zrównoleglenie wykonywania czynności systemu. Również w przypadku bazy danych — jednego z istotniejszych zasobów systemu — dąży się do zapewnienia dostępu współbieżnego.

W pierwszej części artykułu przedstawiamy wybrane aspekty funkcjonowania oprogramowania zarządzającego współbieżną aktualizacją bazy danych. Część druga poświęcona będzie przeglądowi metod leżących u podstaw tworzenia takiego oprogramowania.

Zarządzanie współbieżną aktualizacją bazy danych (ang. concurrent database update control) polega na szeregowaniu odwołań do bazy danych pochodzących od procesów aktualizujących, działających współbieżnie z procesami ją wykorzystującymi. Równoczesny dostęp nie stwarza problemów w przypadku, gdy żaden z procesów współbieżnych nie aktualizuje bazy danych. Operacje odczytu z bazy danych związane z różnymi procesami mogą się wtedy dowolnie przeplatać w czasie. Jeśli jednak choć jeden z procesów współbieżnych dokonuje aktualizacji, to w działaniu systemu mogą wystąpić trojakiego rodzaju nieprawidłowości [3]:

- **Utrata przeprowadzonej aktualizacji bazy danych.**

PRZYKŁAD. We wspólnej bazie danych znajduje się dana x=4 oznaczająca liczbę wolnych miejsc w samolocie. Proces A dokonuje rezerwacji jednego miejsca poprzez zmniejszenie wartości danej x o 1. Proces B działa identycznie. Jeżeli procesy A i B działają współbieżnie, to wtedy może wystąpić następująca sekwencja akcji w dostępie do bazy danych:
— proces A dokonuje odczytu x=4
— proces B dokonuje odczytu x=4
— proces A dokonuje zapisu x=3
— proces B dokonuje zapisu x=3.
W konsekwencji oznacza to utratę informacji o zarezerwowaniu jednego miejsca przez proces A.

- **Dezinformowanie procesu dokonującego odczytów z bazy danych.**

PRZYKŁAD. Proces A dokonuje przelewu z konta x na konto y. Proces B oblicza sumę kont x i y. Akcje należące do tych procesów mogą ułożyć się w kolejności:
— proces B dokonuje odczytu x
— proces A dokonuje wszystkich akcji związanych z przelewem
— proces B dokonuje odczytu y.
W rezultacie takiego uszeregowania akcji, proces B źle obliczył sumę kont x i y.

- **Wprowadzenie błędu do bazy danych w przypadku, gdy dezinformacji ulegnie proces aktualizujący bazę danych.**

Oczywistym rozwiązaniem pozwalającym unikać tych błędów w działaniu systemu jest zezwolenie procesom współbieżnym jedynie na dokonywanie odczytów z bazy danych. Aktualizacja bazy dokonuje się wtedy za pomocą pojedynczego procesu działającego wówczas, gdy system jest niedostępny dla pozostałych użytkowników.

Okresową aktualizację można zastosować — na przykład — w systemie wyszukiwania informacji bibliograficznej, gdzie aktualizowanie bazy danych praktycznie wystarcza raz na miesiąc lub nawet rzadziej. Z kolei systemem w którym akceptuje się 24-godzinne opóźnienie w pojawianiu się uaktualnień w bazie danych jest system zarządzania przedsiębiorstwem.

Okresowe aktualizowanie bazy danych jest natomiast niewystarczające w takich zastosowaniach, jak rezerwacja miejsc lotniczych lub sterowanie ruchem na lotnisku, gdzie wszystkie zmiany muszą być wprowadzane do bazy danych na bieżąco. Oznacza to konieczność zezwolenia na aktualizację bazy danych w warunkach współbieżnej pracy wielu procesów, a to z kolei wymaga stosowania dodatkowych mechanizmów, które system zabezpieczą.

Modelowanie fragmentu rzeczywistości w postaci bazy danych związane jest z przyjęciem pewnych założeń dotyczących danych zawartych w bazie oraz powiązań między tymi danymi. Założenia te nazywa się ograniczeniami spójności (ang. consistency constrains). Przykładem takich ograniczeń może być zdanie: „liczba sprzedanych na dany lot biletów nie może przekraczać liczby miejsc w samolocie". Stan bazy danych, w którym spełnione są wszystkie zależności wynikające z ograniczeń spójności nazywa się **stanem spójnym.**

Program użytkowy, korzystając z bazy danych, realizuje jedną lub kolejno (zwykle cyklicznie) wiele operacji dostępu do bazy danych, zwanych **transakcjami.** Transakcją jest ciąg operacji na bazie danych stanowiący pewną logiczną całość. Wyróżnia się trzy typy transakcji, których przykładami mogą być ciągi operacji związane z realizacją w systemie informatycznym takich poleceń, jak:

- dokonanie przelewu między dwoma kontami
- obliczenie sumy stanu wszystkich kont
- zmiana adresu zamieszkania pracownika.

Przypadek pierwszy ilustruje **transakcję aktualizacji** powodującą zmianę stanu bazy danych, przypadek drugi — **transakcję zapytania** (nie dokonującą żadnego zapisu do bazy danych), a przypadek trzeci — **specyficzną transakcję aktualizacji,** którą można określić mianem **arbitralnej transakcji aktualizacji,** gdyż nowa wartość danej nie zależy od jej dotychczasowej wartości i w związku z tym stara wartość w ogóle nie musi być odczytana przed dokonaniem aktualizacji.

KOORDYNATOR SZBD

Funkcja zarządzania współbieżną aktualizacją bazy danych realizowana jest przez moduł zwany koordynatorem, wchodzący w skład Systemu Zarządzania Bazą Danych (SZBD). Szereguje on wszelkie żądania zapisu i odczytu kierowane do innej części SZBD, zwanej **modułem dostępu,** który te żądania realizuje. Koordynator współpracuje ponadto również z modułem SZBD odpowiedzialnym za odtwarzanie bazy danych. Możliwości tego ostatniego są brane pod uwagę w algorytmach działania koordynatora, gdyż pozwalają odwoływać te podjęte przez niego decyzje, które okazały się błędne (zbyt optymistyczne).

Strumień żądań zapisu i odczytu nadchodzący do koordynatora nazywa się **strumieniem wejściowym** (ang. schedule). W strumieniu tym żądania należące do różnych transakcji przeplatają się, przy czym kolejność żądań należących do jednej transakcji pozostaje zachowana. Fragment strumienia wejściowego przedstawić można następująco:

T_1 O(x) Z(x) O(y) Z(y)
T_2: O(x) Z(x) O(y) Z(y)

gdzie O(x) oznacza żądanie odczytu danej x, Z(x) — żądanie zapisu danej x, a T_1 oznacza i-tą transakcję (i=1,2,...). Od koordynatora wymaga się takiej zmiany w kolejności żądań strumienia wejściowego, aby powstał nowy strumień żądań zwany **strumieniem wyjściowym,** który można przekazać do realizacji modułowi dostępu do bazy danych bez obawy o wystąpienie błędów spowodowanych współbieżną aktualizacją.

Najprostszym, trywialnym rozwiązaniem problemu sterowania współbieżną aktualizacją bazy jest przekształcenie strumienia wejściowego w szeregowy strumień wyjściowy, w którym żadna operacja transakcji nie występuje pomiędzy operacjami innej transakcji. Rozwiązanie to oznacza zupełny brak współbieżności, lecz równocześnie z oczywistych względów eliminuje wszystkie anomalie wynikające ze współbieżności. Poszukując rozwiązań nietrywialnych, które w jak najmniejszym stopniu ograniczają współbieżność, powszechnie przyjmuje się jako kryterium poprawności funkcjonowania koordynatora — generowanie przez niego szeregowalnych strumieni wyjściowych. Przez **strumień szeregowalny** rozumie się strumień równoważny strumieniowi szeregowemu zarówno pod względem jego wpływu na stan bazy danych, jak i na odpowiedzi systemu udzielane w transakcjach zapytań. Można wyróżnić dwie następujące odmiany strumieni szeregowalnych:

- strumienie generujące tylko takie stany bazy danych, jakie mogą powstać w przypadku strumienia szeregowego; w skrócie określa się je nazwą PSV (Preserving Serial View), co oznacza — **zachowujące obraz szeregowy**
- strumienie gwarantujące końcowy stan bazy danych identyczny ze stanem, do którego doprowadziłoby szeregowe wykonanie wszystkich transakcji występujących w strumieniu wejściowym, ale dopuszczające pośrednie stany bazy nie należące do zbioru stanów generowanych przez strumień szeregowy; w skrócie określa się je nazwą SFS (Serial Final State), co oznacza — **szeregowy stan końcowy.**

Istnienie strumieni typu SFS jest możliwe dzięki arbitralnym transakcjom aktualizacji. Wystąpienie takiej transakcji może bowiem spowodować, że niektóre transakcje aktualizacji przestają być transakcjami żywymi, przy czym za transakcję żywą uważa się [2]:

- hipotetyczną transakcję, która przed zamknięciem systemu dokonuje odczytu całej bazy danych
- każdą transakcję, której efekt działania (lub choćby jego część) czytany jest przez transakcję żywą
- każdą transakcję zapytania.

Transakcje nie będące transakcjami żywymi nazywa się **transakcjami martwymi.** Przykładowo — w strumieniu

T_1 : O(x) Z(x)
T_2 : O(x) Z(x)
T_3 : O(y) Z(x)

transakcje T_1 i T_2 są martwe, ponieważ efekt ich działania zamazywany jest przez arbitralną transakcję aktualizacji T_3. W praktyce występowanie transakcji martwych jest rzadkie, ponieważ rzadkie są arbitralne transakcje aktualizacji. W związku z tym, zwiększenie współbieżności w systemie poprzez stosowanie strumieni wyjściowych typu SFS zamiast typu PSV jest na ogół nieistotne. Przyjmuje się też, że chociaż koordynator powinien zapewniać szeregowalność, to można rozważać systemy, w których wyższy stopień współbieżności jest ważniejszy od pełnej ochrony przed anomaliami współbieżnej aktualizacji.

Przykładowo, w [4] rozważano możliwość różnych rodzajów ochrony częściowej — tak, aby transakcja zapytania zawierająca informacje o charakterze statystycznym mogła uzyskiwać bez czekania dostęp do niespójnych danych, jeżeli wynikający stąd błąd jest nieznaczący.

WZNAWIANIE TRANSAKCJI

Działanie koordynatora, bez względu na to jakie zostały w nim użyte mechanizmy, jest w istocie formą rozdziału zasobów wspólnych — jakimi są dane z bazy danych — pomiędzy procesy współbieżne realizujące poszczególne transakcje. Jeżeli znane są wszystkie żądania procesów dotyczące przydziału zasobów, wtedy przydziału można dokonywać w taki sposób, aby nigdy nie zachodziła potrzeba wywłaszczania procesów z przydzielonych im zasobów, spowodowana zakleszczeniem (ang. deadlock).

Podejmowanie decyzji bez pełnej znajomości wszystkich żądań może doprowadzić do konieczności wywłaszczenia procesu i wznowienia transakcji. Wznowienie transakcji może być poprzedzone unieważnieniem (wycofaniem) z bazy danych zmian wprowadzonych w związku z tą transakcją. Do wznawiania transakcji zarówno niezakończonych, jak i już zakończonych używa się modułu odtwarzania bazy danych. Jeżeli koordynator działa tak, że zakłada możliwość wznawiania transakcji już zakończonych, to należy liczyć się z tym, że wznowienie jednej może pociągnąć za sobą wznowienie wielu transakcji. Zjawisko to nosi nazwę efektu domina. Kłopotliwa jest przy tym nie tyle konieczność powtórzenia wielkiej liczby obliczeń, ile na przykład fakt, że użytkownik mógł już odejść od terminala i trudno go zawiadomić, że przeprowadzona przez niego transakcja została unieważniona.

Rozpatrzmy przypadek strumienia (1). Jeżeli dopuszcza się wznawianie transakcji zakończonych, to koordynator może przekazać bezpośrednio do strumienia wyjściowego kolejne elementy strumienia wejściowego aż do momentu nadejścia żądania O(y), należącego do transakcji T_1. W tym momencie staje się jasne, że postępowanie koordynatora było zbyt optymistyczne i należy wznowić transakcję T_2. Zauważmy, że optymistyczne działania koordynatora mogłyby zakończyć się sukcesem, gdyby transakcja T_1 nie zawierała żądania O(y). Jeżeli dopuszcza się wznowienie jedynie transakcji niezakończonych, to w przypadku strumienia (1) koordynator musiałby wstrzymać wykonywanie wszelkich akcji transakcji T_2. Każda z nich mogłaby bowiem okazać się ostatnią akcją tej transakcji.

Znana jest ogólna zasada postępowania koordynatora odbierająca możliwość wystąpienia sytuacji, w której konieczne byłoby wznowienie transakcji zakończonych — **aż do momentu zakończenia transakcji nie należy zezwalać innym transakcjom na odczyt zaktualizowanych przez nią danych.** Zasada ta chroni przed wznawianiem transakcji zakończonych, nie tylko wtedy, gdy wystąpi brak szeregowalności strumienia wyjściowego, ale również w przypadku dowolnej awarii systemu. Fakt ten jest niezwykle istotny. Jeżeli bowiem w danym systemie niedopuszczalne jest wznawianie transakcji zakończonych, to w obawie przed wystąpieniem efektu domina na skutek awarii, nie zezwala się innym transakcjom na odczyt danych zaktualizowanych przez niezakończoną jeszcze transakcję. W systemie takim nie można więc również stosować koordynatorów zakładających większą swobodę w odczycie aktualizowanych danych, nawet jeżeli dany koordynator może zagwarantować brak zakleszczeń i związanych z nimi wznowień transakcji.

Zasadniczo rozważa się jedynie koordynatory kierujące się wyłącznie składnią transakcji, wyrażoną w sekwencji żądań tworzących strumień wejściowy, i nie biorące pod uwagę semantyki transakcji ani semantyki ograniczeń spójności. Jakkolwiek dysponowanie informacją semantyczną pozwalałoby zminimalizować ograniczanie przez koordynator współbieżności w systemie (p. poniższy przykład), to analiza semantyczna jest w ogólnym przypadku praktycznie niewykonalna [7].

Rozpatrzmy prosty przykład uwzględniania semantyki i transakcji. Załóżmy, że ograniczenia spójności wymagają, aby dana x miała tę samą wartość co dana y. Strumień wejściowy (1) w ogólnym przypadku nie może być przekazany na wyjście koordynatora, ponieważ naruszałoby to spójność bazy danych, np. w przypadku, gdy T_1 zmniejsza x i y, a T_2 podwaja ich wartość. Jeżeli jednak zarówno T_1, jak i T_2 podwajają wartość x i y, to rozważany strumień wejściowy nie narusza spójności, a więc koordynator znający semantykę transakcji mógłby ten strumień przekazać modułowi dostępu do bazy danych bez żadnych zmian, czyli opóźnień poszczególnych akcji.

SKŁADNIA TRANSAKCJI

W zagadnieniu tym można wyróżnić cztery przypadki:

- Koordynator poznaje składnię transakcji w miarę napływania kolejnych żądań z nią związanych. Jeżeli koordynator — zanim pozna pełną składnię transakcji — podejmuje decyzje przekazywania żądań do strumienia wyjściowego, może zajść konieczność zmiany tej decyzji i wznowienia transakcji. Przy zachowaniu dużej ostrożności, konieczne wznawianie transakcji ograniczy się do transakcji niezakończonych. Przy „optymistycznym" przekazywaniu żądań do strumienia wyjściowego, może wystąpić konieczność wznawiania również transakcji zakończonych.
- Koordynator posiada częściowe informacje o składni każdej rozpoczynającej się transakcji. Jedną z możliwości jest znajomość klasy, do jakiej należy dana transakcja (podejście takie zastosowano w systemie rozproszonej bazy danych SDD-1 [1]). Jeżeli w strumieniu wejściowym pojawią się transakcje z klas nie kolidujących z sobą, działających na różnych grupach danych, koordynator może przekazywać je do strumienia wyjściowego bez obawy wystąpienia konieczności wznawiania jakichkolwiek transakcji. Ciągle musi być jednak zachowana ostrożność w przypad-

ku transakcji należących do klas kolidujących z sobą, przy czym oczywiście może się okazać, że konkretne transakcje w istocie nie operowały na tych samych danych i można było uzyskać większą współbieżność.

Innego typu informacją dotyczącą struktury transakcji może być częściowe uszeregowanie danych i narzucenie programom użytkowym obowiązku żądania danych w ustalonej kolejności. W rezultacie takiej częściowej informacji o strukturze transakcji [5, 6] uniknięto niebezpieczeństwa wznowień transakcji w koordynatorze posługującym się techniką blokowania (p. druga część artykułu).

Przykładem częściowej informacji o składni transakcji jest — także przyjęcie modelu transakcji złożonej z dwóch części [2]: w części pierwszej dokonuje się odczytów danych, natomiast w drugiej — zapisów danych. Ograniczenie to pozwoliło zdefiniować transakcje słabo dwufazowe.

- Koordynator zna dokładnie składnię każdej rozpoczynającej się transakcji, co oznacza, że transakcje zgłaszają z góry zapotrzebowanie na wszystkie dane, które będą przez nie wykorzystywane. W ogólnym przypadku jest to jednak niemożliwe, ponieważ czasem można powiedzieć, jakie dane są jeszcze potrzebne do zrealizowania transakcji, dopiero po przeanalizowaniu informacji uzyskanych przez odczyt pewnych danych z bazy danych. Zauważmy, że zgłaszanie zapotrzebowania na wyrost (na wszelkie potencjalnie potrzebne dane) sprowadza ten przypadek do poprzedniego.
- Koordynator zna z góry wszystkie transakcje, jakie w pewnym czasie będą przeprowadzane w systemie i zna dokładnie ich składnię. Jakkolwiek trudno podać przykład systemu, w którym można by urzeczywistnić taką sytuację, z teoretycznego punktu widzenia przypadek jest interesujący i był rozważany w [2].

☆ ☆ ☆

W literaturze przyjmuje się, że algorytm przekształcania strumienia wejściowego w strumień wyjściowy musi mieć złożoność wielomianową i dlatego odrzuca się rozwiązania o większej złożoności — jako powodujące zbyt duży narzut czasowy, uniemożliwiający praktycznie pracę w czasie rzeczywistym.

Z rozważań dotyczących funkcji koordynatorów wynika podstawowy wniosek, iż porównywanie koordynatorów ma sens jedynie w ramach koordynatorów należących do tej samej klasy, a więc posiadających taki sam zasób informacji o semantyce i składni transakcji oraz takie same uprawnienia co do wznawiania transakcji niezakończonych i zakończonych oraz gwarantujących takie same efekty, jak na przykład szeregowalność strumienia wyjściowego. Jedynie w ramach danej klasy można mówić o rzeczywistym porównywaniu dwóch koordynatorów, przy czym za lepszy należałoby uznać koordynator wprowadzający mniej opóźnień w systemie.

Na wielkość opóźnień wpływa przede wszystkim złożoność czasowa algorytmu (złożoność pamięciowa raczej nie jest istotna) i wielkość zbioru charakterystycznego (ang. fixed point [8, 9]), czyli procent strumienia wejściowego przepuszczanego przez koordynator bez żadnych opóźnień do strumienia wyjściowego, a także inne cechy, jak — na przykład — narzut związany ze wznawianiem transakcji wymuszanym przez koordynatora. Uświadomienie sobie powyższego faktu dotyczącego klas koordynatorów pozwala we właściwy sposób widzieć wyniki porównań koordynatorów przeprowadzanych choćby według wielkości zbioru charakterystycznego, a obejmujących koordynatory należące do różnych klas.

LITERATURA

[1] Bernstein P. A., Shipman D. W., Rothnie J. B.: Concurrency control in a System for Distributed Databases (SDD-1). ACM Trans. on DBS, vol. 5, No. 1, March 1980, pp. 18—51
[2] Bernstein P. A., Shipman D. W., Wong W. S.: Formal aspects of serializability in database concurrency control. IEEE Trans. on Soft. Eng., vol. SE-5, No. 3, May 1979, pp. 204—216
[3] Casanova M. A., Bernstein P. A.: General Purpose Schedulers for Database Systems. Acta Inf., vol. 14, 1980, pp. 195—220
[4] Gray J. N., Lorie R. A., Putzolu G. R., Traiger I. L.: Granularity of locks and degrees of consistency in a shared data base. Modeling in Data Base Systems, G. M. Nijssen (ed), North Holland Publ. Co., 1976
[5] Kedem Z. M.: Locking protocols: from exclusive to share locks. Journal of the ACM 30 (4), 1983, pp. 787—804
[6] Kedem Z. M., Silberschatz A.: A characterization of database graphs admitting a simple locking protocol. Acta Inf., 1981 (16), pp. 1—13
[7] Kung H. T., Papadimitriou C. H.: An optimality theory of concurrency control for database. Acta Inf., 1983, pp. 1—11
[8] Papadimitriou C. H.: A theorem in database concurrency control. Journal of the ACM, 29 (4), 1982, pp. 998—1006
[9] Papadimitriou C. H.: The serializability of concurrent database updates. J. Ass. Comp. Mach. 26 (4), 1979, pp. 631—653.

WACŁAW ISZKOWSKI
Instytut Informatyki
Politechnika Warszawska

Przegląd mechanizmów komunikacji

Synchronizacja współdziałania procesów działających współbieżnie może być realizowana z wykorzystaniem mechanizmów kooperacji lub komunikacji. Mechanizm kooperacji polega na wprowadzaniu wzajemnego wykluczania w wykonywaniu określonej grupy procedur wielodostępnych; tworząc monitor będący zestawem takich procedur, nakłada się warunek, że tylko jedna z nich może być w danej chwili wykonywana. Uproszczoną odmianą tego mechanizmu jest mechanizm synchronizacji z wykorzystaniem operacji semaforowych [4]. Mechanizm komunikacji polega zaś na przesyłaniu wiadomości (ang. message) między parami procesów — wprowadza wzajemną ich synchronizację na etapie nadawania i odbierania informacji. Mechanizm ten ma przede wszystkim zastosowanie w rozproszonych systemach wieloprocesorowych, chociaż wykorzystuje się go też w implementacjach logicznie wieloprocesowych systemów jednoprocesorowych.

Oba mechanizmy — kooperacji i komunikacji — są strukturalnie izomorficzne, a więc jest możliwe przekształcenie systemu procesów kooperujących w system procesów komunikujących się i odwrotnie (przy pewnych ograniczeniach [6]). Historycznie, mechanizm komunikacji był pierwszy — biorąc pod uwagę komunikację programów z otoczeniem. Później znajdował jedynie zastosowanie w sieciach telekomunikacyjnych. Teraz — po dopracowaniu — stał się mechanizmem wiodącym.

Istnieje wiele koncepcji zapisu i realizacji komunikacji procesów. Przytaczanie wszystkich spotykanych rozwiązań byłoby niecelowe. Dlatego też przedstawiam kilka wyróżniających się cech charakterystycznych w najczęściej spotykanych wersjach. Są to:
- operacje przesyłania wiadomości
- powiązanie nadawcy z odbiorcą
- formy powiązań krotnych
- buforowanie wiadomości.

Przedstawiam najpierw przegląd spotykanych rozwiązań, by następnie zaproponować zestaw, który klasyfikuje mechanizm komunikacji według cech charakterystycznych. Wykorzystując tę klasyfikację, opiszę dla przykładu trzy znane mechanizmy komunikacji, zastosowane w systemie RC 4000, procesach komunikacyjnych Hoare'a oraz zdefiniowane w języku ADA.

Mechanizmy komunikacji powinny spełniać następujące warunki [7]:

- **rozproszenie**, rozumiane jako zdolność adaptacji mechanizmu do dynamicznie rekonfigurowanych systemów wieloprocesorowych — aby możliwe było działanie tylko części systemu na różnych procesorach
- **efektywność**, określana stopniem ograniczeń w rozproszeniu systemu, umożliwiających sprawną komunikację między procesami (porównywalna z czasem przełączania wywoływanych procedur w językach wysokiego poziomu)
- **prostota**, wyznaczona łatwością użycia oraz możliwościami łatwej implementacji w rozpatrywanym systemie
- **protekcja**, uwzględniająca odporność na błędne działanie fragmentu systemu, bez wpływu tego uszkodzenia na poprawne działanie pozostałej części systemu.

Podane jakościowe warunki sprawności działania mechanizmu komunikacji są trudne do ilościowego określenia — tym bardziej, że pożądane cele są wzajemnie uzależnione.

OPERACJE PRZESYŁANIA WIADOMOŚCI

W parze procesów synchronizowanych z wykorzystaniem mechanizmu komunikacji, jeden z nich jest procesem nadawcy (ang. sender), a drugi — procesem odbiorcy (ang. receiver) wiadomości. Połączenie między nadawcą a odbiorcą jest nazywane potokiem (ang. pipe), który przekazuje przesyłaną wiadomość, specyfikując sposób identyfikacji nadawcy i odbiorcy.

W podstawowym zbiorze operacji realizujących przesyłanie wiadomości wyróżniamy operacje:
wyślij wiadomość **do** potoku;

— wysłania (ang. send) wiadomości do odbiorcy określonego identyfikatorem potoku, przy czym proces wydający tę operację jest wstrzymywany do momentu możliwej realizacji oddania treści wiadomości w potok

odbierz wiadomość **z** potoku;

— odebrania (ang. receive) wiadomości, lokalizowanej następnie w obiekcie identyfikowanym jako „wiadomość", od nadawcy identyfikowanego przez potok, przy czym proces wydający tę operację jest wstrzymywany do momentu uzyskania wiadomości z potoku

zwróć odpowiedź **do** potoku;

— zwrotu (ang. return, reply) odpowiedzi przez proces odbiorcy do procesu nadawcy określonego przez potok

oczekuj odpowiedź **z** potoku;

— oczekiwania (ang. wait, await) w stanie wstrzymanym procesu nadawcy wiadomości na odpowiedź lokalizowaną w obiekcie „odpowiedź" od procesu odbiorcy identyfikowanego przez potok

wyślij wiadomość **do** potoku **z** odpowiedzią;

— łączącej cechy operacji wyślij i oczekuj.

Podane w zestawie operacje w różnych implementacjach mogą być zapisywane w odmiennych postaciach, zachowując swe zasadnicze cechy. W praktyce spotykamy również inne operacje, a w wielu przypadkach operacje przekazywania odpowiedzi nie są realizowane.

Zależnie od implementacji wiadomością identyfikowaną tutaj jako „wiadomość" może być zbiór wielu wartości lub struktura danych je przechowująca, będąca rekordem danych o stałej lub zmiennej długości i strukturze. Zawiera ona wartości danych informacyjnych lub referencje do obiektów przekazywanych do procesu odbiorcy, a niekiedy

— identyfikator procesu nadawcy. W niektórych przypadkach wiadomość może nie mieć zawartości i zwana jest wtedy sygnałem (ang. signal). Oprócz tego możliwa jest interpretacja i reorganizacja wiadomości przez potok. W takim przypadku sekwencja wiadomości wysłanych przez nadawcę może stać się wiadomością dla odbiorcy pojedynczą, bez rozróżniania jej składowych. W praktyce istnieje tutaj duża dowolność, zależna od przyjętej struktury systemu zarządzającego przesyłaniem wiadomości.

Uzupełnieniem operacji przesyłania wiadomości może być faza czynności czasowego ograniczania (ang. timeout) wykonywania operacji wysłania lub odbioru wiadomości. Dołączenie do zapisu operacji frazy **w czasie** wyrażenie nakłada dodatkowy warunek, aby realizacja operacji zakończyła się przed upływem podanej wyrażeniem liczby jednostek czasu. W przeciwnym razie operacja jest traktowana jako pusta i proces ją generujący może działać dalej. Innymi słowy fraza ta tworzy klasę uwarunkowanych czasowo operacji przesyłania wiadomości.

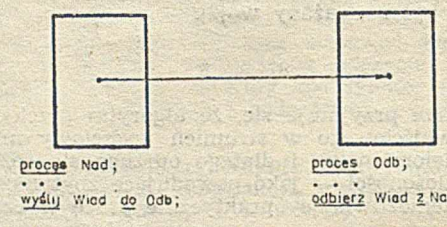
POWIĄZANIE NADAWCY Z ODBIORCĄ

Istotnym elementem rozróżniającym poszczególne mechanizmy komunikacji jest metoda identyfikacji procesu odbiorcy u nadawcy, a procesu nadawcy u odbiorcy.

Dokonuje się jej przez identyfikację potoku. Wyróżniamy tutaj cztery podstawowe metody:

A. Bezpośrednia identyfikacja procesów

W procesie nadawcy identyfikatorem potoku jest identyfikator procesu odbiorcy, a w procesie odbiorcy — identyfikator nadawcy. Powiązanie to jest statyczne i ewentualna zmiana liczby wspólnie działających procesów w systemie powoduje konieczność przeprogramowania niektórych z nich. Wzajemna znajomość identyfikatorów procesów eliminuje potrzebę umieszczania identyfikatora nadawcy w treści wiadomości. Bezpośrednie powiązanie procesów pokazano schematycznie na rysunku 1.

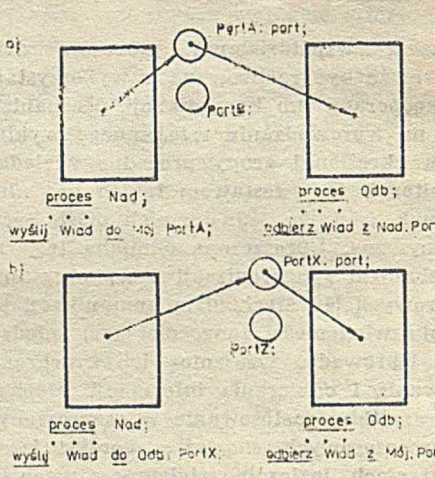


Rys. 1. Powiązanie przez bezpośrednią identyfikację procesów

B. Bezpośrednia identyfikacja portów procesów

Portem (ang. port) procesu jest wyróżniony obiekt związany z danym procesem i określający miejsce wprowadzenia lub wyprowadzenia wiadomości. Wyróżniamy więc tutaj odpowiednio porty wejściowe procesu oraz porty wyjściowe, przy czym niektóre z nich lub wszystkie mogą pełnić obie funkcje. Każdy proces może mieć dowolną liczbę portów.

Do portu wyjściowego procesu nadawcy ma bezpośredni dostęp każdy proces odbiorcy, przy czym identyfikator tego procesu nie jest znany nadawcy. I analogicznie — do portu wejściowego procesu odbiorcy ma dostęp każdy proces nadawcy, przy czym jego identyfikator nie jest znany odbiorcy. W praktycznej realizacji wykorzystywane są powiązania: **nadawca — jego port wyjściowy — odbiorca** oraz **nadawca — port wejściowy odbiorcy-odbiorca.** Powiązanie:



Rys. 2. Powiązanie procesów przez bezpośrednią identyfikację portów procesów a) u nadawcy, b) u odbiorcy

nadawca — jego port wyjściow — port wejściowy odbiorcy — odbiorca jest możliwe jedynie do zastosowania przy wykorzystaniu kanału (p. pkt C).
Opisane powiązania są statyczne, z tym że ewentualna zmiana niektórych z procesów pociąga za sobą konieczność przeprogramowania tylko niektórych procesów. Schematy powiązania procesów przez porty przedstawia rysunek 2.

C. Powiązanie procesów przez kanał

Kanałem (ang. channel, link) jest wyróżniony obiekt globalny systemu procesów pełniący rolę potoku o znanym dla nadawcy i odbiorcy identyfikatorze. Kanały mogą być niezależne od procesów lub też mogą być statycznie lub dynamicznie przydzielane określonym klasom procesów. Możliwe jest też dynamiczne tworzenie kanału na żądanie procesu, któremu jest on wtedy dedykowany.

Dopuszczenie istnienia w programie współbieżnym niezależnych kanałów umożliwia prostą modyfikację programu poprzez usuwanie lub tworzenie procesów — bez potrzeby przeprogramowywania pozostałych. Procesy nadawcy i odbiorcy nie znają wzajemnie swoich identyfikatorów, a więc konieczne może być przekazywanie wiadomości identyfikatora procesu nadawcy, a w odpowiedzi — identyfikatora procesu odbiorcy.

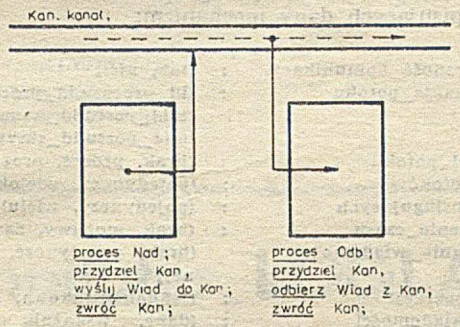
Dedykując kanały procesom, korzystamy ze statycznej deklaracji przypisującej kanał procesowi lub z operacji:

utwórz kanał;
— tworzenie (ang. create) kanału w danym procesie,

przydziel kanał;

zwróć kanał;

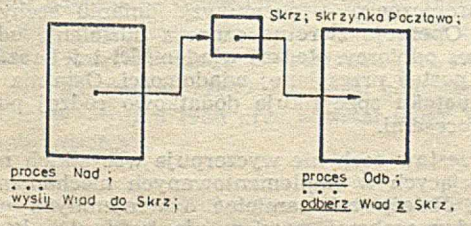
— przydziału (ang. allocate) i zwrotu (ang. return, deallocate) kanału dla procesu.
Schemat powiązania procesów przez kanał przedstawia rysunek 3.



Rys. 3. Powiązanie procesów przez kanał

D. Powiązanie procesów poprzez skrzynkę pocztową

Skrzynka pocztowa (ang. mailbox) jest wyróżnionym obiektem globalnym dla systemu procesów, pełniącym rolę potoku o znanym identyfikatorze. Skrzynki pocztowe są niezależne od procesów i występują niezależnie od istnienia procesów. Tym samym dynamiczna zmiana części struktury powiązań i liczby procesów nie ma wpływu na pozostałą część systemu. Proces nadawcy składa wiadomość w skrzynce, z której odbiera ją proces odbiorcy. Procesy nie znają wzajemnie swoich identyfikatorów. W pewnych przypadkach, skrzynki pocztowe mogą być logicznie utożsamiane z kanałami, przy czym ich fizyczna realizacja jest odmienna i wynika z architektury systemu wieloprocesowego. Schemat powiązania procesów przez skrzynki obrazuje rysunek 4.



Rys. 4. Powiązanie procesów przez skrzynkę pocztową

Podane metody powiązania procesów są dla mechanizmu komunikacji jednorodne i symetryczne. W praktycznych zastosowaniach spotyka się ich różne odmiany, łączące w sobie cechy dwóch lub więcej metod.

Dodatkowym elementem zwiększającym efektywność mechanizmów komunikacji procesów jest dopuszczenie współpracy z danym potokiem wielu procesów nadawców i odbiorców, a także dopuszczenie do równoczesnego wysyłania wiadomości przez wiele potoków oraz oczekiwania na wiadomość pochodzącą z jednego z wielu potoków.

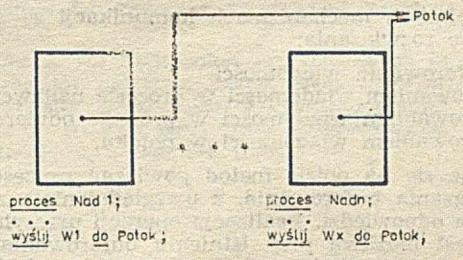
Podane formy współpracy można podzielić na cztery klasy:

- **Jeden proces — jeden potok.**

Każdy proces może wysyłać lub odbierać wiadomość tylko przez jeden potok, wykorzystując jedną z podanych wyżej metod powiązań.

- **Wiele procesów — jeden potok.**

Wiele procesów nadawców może wysyłać wiadomość korzystając z tego samego potoku oraz wiele procesów odbiorców może równocześnie oczekiwać na wiadomość z tego samego potoku. Schemat takiego powiązania dla procesów nadawców przedstawia rysunek 5. Ta forma współpracy nie może istnieć w procesie nadawców dla powiązania procesów przez porty, jeżeli są one u nich zlokalizowane. Analogiczne ograniczenie występuje dla procesów odbiorców.



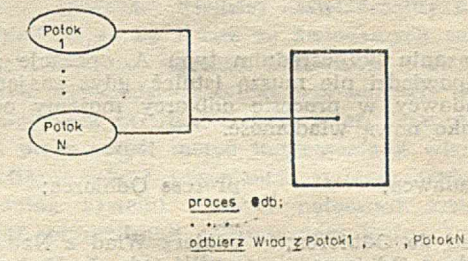
Rys. 5. Wielu nadawców — wspólny potok

- **Jeden proces — wiele potoków.**

Każdy proces może wysyłać tę samą wiadomość przez wiele potoków jednocześnie lub oczekiwać na wiadomość z jednego z wielu potoków. Wyróżniony zbiór potoków określimy tutaj jako:

zbiór_potoków = **set of** potok;

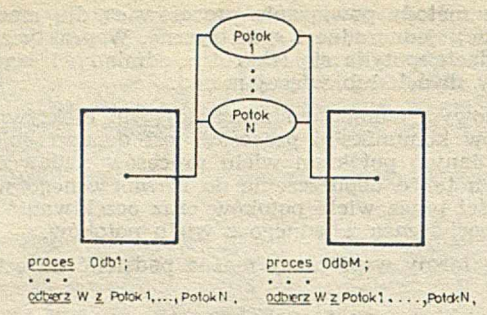
Proces nadawcy prześle wiadomość przez pierwszy z potoków, który to umożliwia, a proces odbiorcy odbierze wiadomość z pierwszego potoku, który ją dostarczy. Kolejność wyboru potoku, spełniającego dany warunek, jest z reguły nieokreślona (jest zależna od implementacji systemu). Schemat takiego powiązania dla procesu odbiorcy przedstawia rysunek 6.



Rys. 6. Jeden proces odbiorcy — wiele potoków

- **Wiele procesów — wiele potoków.**

Klasa ta jest złożeniem dwóch poprzednich klas i umożliwia współpracę wielu procesów z jednym lub z kilkoma potokami ze zbioru potoków. Schemat takiego powiązania dla procesów odbiorców przedstawia rysunek 7.



Rys. 7. Wiele procesów odbiorcy — wiele potoków

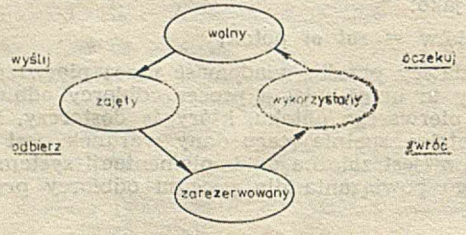
BUFOROWANIE WIADOMOŚCI

Proces nadawcy realizujący operację **wyślij** jest wstrzymywany, jeżeli wiadomość nie może być wysłana. Analogicznie proces odbiorcy jest wstrzymywany oczekując na wiadomość. Takie wstrzymywanie procesów zmniejsza efektywność współbieżności, sprowadzając ją w warunkach ekstremalnych do wykonywania obu procesów jako współprogramów. Dla zwiększenia efektywności, do potoku wiążącego proces nadawcy z odbiorcą wprowadza się obszar buforowy przechowujący nie odebrane wiadomości. Bufor ten przyspiesza działanie procesu nadawcy oraz dostarcza odbiorcy wiadomości, zmniejszając wzajemną ich zależność.

W istniejących mechanizmach komunikacji są stosowane następujące rozwiązania:

— bez buforowania wiadomości
— z buforowaniem wiadomości w procesie nadawcy
— z buforowaniem wiadomości w procesie odbiorcy
— z buforowaniem wiadomości w potoku.

Opierając się na opisie metod powiązań procesów, omówię rozwiązania buforowania, z uwzględnieniem zwrotnego przesyłania odpowiedzi. Realizacja operacji przesyłania wiadomości jest możliwa przy istnieniu buforowania. W tym przypadku każda wiadomość, nawet przy oczekującym na nią odbiorcy, jest wiązana z buforem, który jest rezerwowany po pobraniu wiadomości na przechowywanie odpowiedzi. Bufor ten jest zwalniany po odebraniu odpowiedzi przez proces nadawcy. Stąd też każdy taki element bufora znajduje się w stanach [1] pokazanych na rysunku 8.



Rys. 8. Graf zmian stanów bufora

W powiązaniu bezpośrednim typu A, operacje ze zwracaniem odpowiedzi nie muszą istnieć, gdyż znając identyfikator nadawcy w procesie odbiorcy możemy odpowiedź przesłać jako nową wiadomość.

process Nadawca;	**process** Odbiorca;
.....	.....
wyślij Wiad **do** Odbiorca;	**odbierz** Wiad **z** Nadawca;
odbierz Odp **z** Odbiorca;	**wyślij** Odp **do** Nadawca;
.....	.....

W powiązaniu bezpośrednim typu B, przez porty, buforowanie może być jedynie związane z portami procesu. Udzielanie odpowiedzi przy lokalizacji portów w procesie odbiorcy (nadawca jest wtedy nieznany) wymaga wykorzystania elementu bufora, zgodnie z podanym wyżej opisem, lub też uzyskania identyfikatora nadawcy z treści przesłanej wiadomości.

W powiązaniu pośrednim typu C, identyfikator nadawcy może być uzyskany z wiadomości, a dla przesłania odpowiedzi kanał musi być dwukierunkowy (musi to być jego cecha fizyczna). Istnienie bufora (co najmniej jednoelementowego) w kanale, przy globalnym przydziale kanałów procesom, utożsamia logicznie to powiązanie z powiązaniem przez skrzynki pocztowe.

W powiązaniu pośrednim typu D, z wykorzystaniem skrzynek, może istnieć tylko buforowanie globalne, gdzie elementy bufora są skrzynkami pocztowymi.

W praktycznej realizacji należy pamiętać, że kopiowanie wiadomości do bufora z procesu nadawcy, a następnie z bufora do dawno już oczekującego procesu odbiorcy, jest stratą czasu. Istnieje więc tutaj pole do optymalizacji tego postępowania. Uwzględniając opisane wyżej zasady, można wprowadzić tu definicję synchronicznego i asynchronicznego mechanizmu komunikacji.

Synchronicznym mechanizmem komunikacji jest przesyłanie wiadomości bez udzielania odpowiedzi i bez buforowania w potoku lub przesyłanie z udzielaniem odpowiedzi i ewentualnym buforowaniem. Mechanizm synchroniczny zapewnia, że para procesów (nadawca-odbiorca) będzie wykonywać w tym samym czasie operacje logiczne sprzężone ze sobą. Asynchronicznym mechanizmem komunikacji jest przesyłanie wiadomości bez udzielania odpowiedzi z wykorzystaniem buforowania w potoku. W tym przypadku operacja wysłania wiadomości może być zrealizowana znacznie wcześniej od momentu realizacji operacji odebrania wiadomości.

OPIS MECHANIZMU KOMUNIKACJI

Różnorodne mechanizmy komunikacji mają kilka cech wspólnych, określających podstawowe zasady ich funkcjonowania. Cechy te mogą być zgrupowane w następujących klasach opisujących dany mechanizm:

a) symetryczność_komunikacji	:	(tak, nie);
b) identyfikacja_potoku	:	(id_procesu,id_procesu.id_portu,id_portu,id_kanału,id_kanału.id_portu,id_skrzynki);
c) właściciel_potoku	:	(brak, proces, program);
d) liczba_potoków	:	(pojedyncza, wielokrotna);
e) liczba_obsługujących	:	(pojedynczy, wielu);
f) ograniczenie_czasu	:	(brak, możliwy, zawsze);
g) buforowanie_wiadomości		(brak, pojedyncze, wielokrotne);
h) rodzaj_operacji	:	(jednokierunkowa, zwrotna);
i) rodzaj_wiadomości	:	(dane, wskaźnik_na_dane,dane_wskaźnik);
j) rozdzielczość_wiadomości	:	(brak, może_istnieć, istnieje);
k) przydział_potoku	:	(statyczny, dynamiczny);

Symetryczność mechanizmu **komunikacji** określa wzajemną jednorodność powiązania procesu nadawcy i odbiorcy; może być w wypadku jej braku rozróżniona cechami określonymi w punktach b—g. **Identyfikacja potoku** specyfikuje rodzaj powiązania opisany jednym z wariantów powiązań typu A, B, C, D. Cecha **właściciel_potoku** precyzuje obiekt zarządzający potokiem, a więc — na przykład — właścicielem kanału może być system lub jeden z procesów, natomiast skrzynki pocztowej — tylko system. W wypadku bezpośredniej identyfikacji potoku przez identyfikator procesu, nie można wskazać jej właściciela. **Liczba potoków** oraz **liczba obsługujących** (ang. server) opisuje jedną z czterech wymienionych poprzednio klas — relacji wzajemnej zbioru procesów ze zbiorem potoków. **Ograniczenie czasowe** (ang. timeout) precyzuje możliwość wykorzystania warunkowych operacji przesyłania wiadomości. Analogicznie, **buforowanie wiadomości** rozróżnia mechanizm synchroniczny od asynchronicznego — przy uwzględnianiu cechy **rodzaj_operacji.** Operacja zwrotna oznacza udzielenie odpowiedzi przez proces odbiorcy. **Rodzaj wiadomości i jej rozdzielczość** precyzuje postać przesyłanej wiadomości. Ostatnia cecha — **przydział potoku** specyfikuje dodatkowo rodzaj powiązania między procesami.

Podany zestaw cech nie wyczerpuje wszystkich możliwości występujących w implementowanych mechanizmach komunikacji. Należy też pamiętać o wzajemnych uzależnieniach między cechami, bowiem nie każda ich kombinacja jest logicznie poprawna.

Wykorzystanie opisu mechanizmu komunikacji dla mechanizmów rzeczywistych

Cechy opisujące mechanizm	Mechanizmy			
	RC4000	CSP	ADA	
			A	B
symetryczność_komunikacji	tak	tak	nie	
identyfikacja_potoku	id_skrzynki	id_procesu	id_procesu.id_portu	id_portu
właściciel_potoku	brak	proces	brak	proces
liczba_potoków	pojedyncza	pojedyncze, wielokrotne	pojedyncze (nadawca)	wielokrotne (odbiorca)
liczba_obsługujących	pojedynczy	pojedynczy	wielu	pojedynczy
ograniczenie_czasu	brak	brak	możliwe	możliwe
buforowanie_wiadomości	wielokrotne	brak	brak	brak
rodzaj_operacji	zwrotna	jednokierunkowa	zwrotna	
rodzaj_wiadomości	dane	dane	dane	
rozdzielczość_wiadomości	brak	może_istnieć	istnieje	
przydział_potoku	statyczny	statyczny	statyczny	

W tabeli przedstawiono wykorzystanie opisu mechanizmu komunikacji dla trzech różnych rzeczywistych mechanizmów. Pierwszy z nich został zrealizowany w systemie RC 4000 i opisany przez Brinch Hansena [1]. Mechanizm ten wykorzystuje skrzynki pocztowe do przesyłania wiadomości oraz zwrotnie — odpowiedzi. Drugi mechanizm jest już klasycznym mechanizmem procesów komunikacyjnych (ang. Communicating Sequential Processes — CSP), zdefiniowanym przez Hoare'a [4] i będącym podstawą wielu późniejszych realizacji (np. w języku BASIC [2, 5]). Ostatni opis przedstawia mechanizm komunikacji włączony do języka ADA [3, 8], zwany popularnie mechanizmem rendez-vous.

LITERATURA

[1] Brinch Hansen P.: Podstawy systemów operacyjnych. WNT, 1979

[2] Draft American National Standard for BASIC. Report ANSI X3J2/82-17, October 1982

[3] Ichbiach J. D. et al.: Reference Manual for the Ada Programming Language, January 1983

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

[5] Iszkowski W., Maniecki M.: Standaryzacja języka BASIC. INFORMATYKA 5, 6, 7—8/1983

[6] Lauer H. C., Needham R. M.: On the duality of operating system structures. In: Operating Systems, ed. Lanciaux D., North Holland, 1979

[7] Ousterhost J. K.: Partitioning and cooperation in a distributed multiprocessor operating system Medusa. Report, Carnegie-Mellon University, Departement of Computer Science, April 1980

[8] Zalewski J.: ADA — nowy język programowania. INFORMATYKA nr 11—12/1981, 1, 2—3, 4—5, 6/1982.

ANDRZEJ I. LITWINIUK
Instytut Informatyki
Uniwersytet Warszawski

Kompilator LOGLANU 82 dla MERY 400

W Instytucie Informatyki UW opracowano w ostatnich latach programy uzupełniające i rozszerzające ubogie — jak dotąd — oprogramowanie podstawowe minikomputera MERA 400. Opracowane programy swoim standardem, wygodą korzystania i parametrami eksploatacyjnymi korzystnie odbiegają od poziomu firmowych produktów. Bardzo ubogi repertuar dostępnych na MERZE języków programowania został rozszerzony między innymi o interpretery PROLOGU i LISPU oraz kompilatory PASCALA i LOGLANU. Poniżej przedstawiam ten ostatni. Zasługuje on na uwagę ze względu na walory samego — niedostatecznie jeszcze rozpowszechnionego — języka LOGLAN, a także dlatego, że jest to jedyna na razie implementacja tego języka. Sam LOGLAN był już prezentowany na łamach INFORMATYKI [1]) [4].

W drugiej połowie lat siedemdziesiątych w Instytucie Informatyki UW prowadzono prace studialne nad uniwersalnym językiem programowania „przyszłościowych maszyn cyfrowych". Był to niezamierzenie — polski odpowiednik programu ADA. W wyniku tych prac powstał nowy język programowania LOGLAN.

Doświadczenia z pierwszej, eksperymentalnej implementacji interpretera zrealizowanego w PASCALU, w systemie CYBER 72, doprowadziły do wprowadzenia wielu zmian do języka LOGLAN. Tę nową wersję nazwano LOGLAN 78. Ukoronowaniem pracy było uruchomienie w końcu 1981 roku kompilatora języka LOGLAN 78 (pod systemem operacyjnym SOM-3) na minikomputerze MERA 400. Kompilator ten miał charakter bardziej doświadczalny niż komercyjny. Wynikało to po trosze ze świadomych samoograniczeń zespołu implementującego, a po trosze z uwarunkowań sprzętowych. Dominującym celem przy budowie tego pierwszego kompilatora LOGLANU było zweryfikowanie wypracowanych przez zespół metod implementacji wielopoziomowego prefiksowania i organizacji systemu wykonawczego (ang. running system). Celowo natomiast zrezygnowano z globalnej optymalizacji generowanego kodu.

Na niekomercyjny charakter kompilatora wpłynęły przede wszystkim jego parametry eksploatacyjne — głównie rozmiar samego kompilatora, a częściowo także niedostateczna szybkość kompilacji. Korzystanie z kompilatora wymagało co najmniej 40 K słów pamięci, z czego sam kod kompilatora zajmował 32 K słów, natomiast resztę — struktury danych. Warto tu wspomnieć, że w typowej konfiguracji MERY 400 programista dysponował jedynie obszarem 20 K słów.

[1]) Prace nad językiem LOGLAN i jego kompilatorem były częściowo finansowane przez Zjednoczenie MERA w ramach problemu węzłowego

Ze względu na stopień skomplikowania języka, kompilator musiał być zrealizowany wieloprzebiegowo. Został on napisany w języku FORTRAN IV w postaci łańcucha ośmiu kolejno wywołujących się programów. Podział na tyle przebiegów wynikał z niskiej jakości kodu generowanego przez dostępny kompilator FORTRANU oraz z ograniczeń pamięciowych narzuconych przez system SOM-3. Logicznie niezbędne były jedynie trzy przebiegi.

Walka z tymi, częściowo sztucznymi, ograniczeniami pamięci istotnie utrudniała pracę zespołu implementującego LOGLAN i raczej negatywnie wpłynęła na ostateczną budowę kompilatora. Trzeba tu przypomnieć, że w 16-bitowej architekturze MERY 400 możliwe jest zaadresowanie 64 K słów, lecz systemowy program scalający ogranicza adresowalną przestrzeń do 32 K słów. To sztuczne ograniczenie przełamano w ten sposób, że z widzialnej przez program scalający przestrzeni adresowej usunięto praktycznie wszystkie struktury danych kompilatora. Struktury te zostały umieszczone w jednym bloku wspólnym, a dodatkowy, specjalnie napisany program przetwarzał produkowany przez kompilator FORTRANU kod wynikowy do postaci, w której odwołania do tego bloku wspólnego zastąpione zostały przez adresy stałe. Ostatecznie, struktury danych kompilatora rezydowały w przestrzeni od 32 K w górę, podczas gdy w dolnej części pamięci umieszczane były kolejne przebiegi. Podany uprzednio rozmiar kodu (32 K słów) dotyczył największych przebiegów: łączna długość kompilatora była kilkakrotnie większa.

Równocześnie z pracą nad implementacją trwały prace nad rozszerzeniami języka. Rozwijany był również sam kompilator, powstało kilka jego kolejnych wersji — aż do obecnie opisywanej.

Podsumowaniem tego etapu prac było opublikowanie raportu języka LOGLAN 82. Różni się on od poprzedniej wersji między innymi wprowadzeniem obsługi wyjątków (ang. exception handling), obsługi plików, stałych definiowanych przez wyrażenia, instrukcji **case** i **repeat**, operatorów **orif** i **andif**, możliwości wstawek w asemblerze oraz procedur i funkcji standardowych [2, 4, 5].

Na MERZE 400 zaimplementowany został cały język LOGLAN 82 z wyjątkiem współbieżności, niezależnej kompilacji i pewnych operacji na plikach. Ograniczenia te wynikały z możliwości systemu operacyjnego (nie dotyczy to niezależnej kompilacji). Kompilator działa pod nieco zmodyfikowanym systemem operacyjnym SOM-3 [3].

BUDOWA KOMPILATORA

Prezentowany kompilator, w porównaniu z pierwszą wersją, został napisany niemal całkowicie na nowo, przy zachowaniu jednak jego oryginalnej struktury. Kompilator został prawie w całości zakodowany w asemblerze GASS [1]. Było to żmudne, lecz niezbędne, gdyż tworzona równolegle wersja zrealizowana w FORTRANIE znacznie przekraczała długością ograniczenia systemowe. Ta benedyktyńska praca przyniosła natomiast korzyść w postaci znacznego przyspieszenia funkcjonowania kompilatora i blisko trzykrotnego jego skrócenia. Kompilator tłumaczy program w LOGLANIE na tekst w asemblerze GASS — każdy moduł programu w LOGLANIE zamieniany jest na oddzielny moduł zapisany w tym asemblerze.

Kompilator jest podzielony logicznie na cztery przebiegi (rozbite łącznie na osiem programów — nakładek). Kod największej z nich ma 13 K słów. Kolejne przebiegi kompilatora realizują następujące zadania:

- analiza syntaktyczna i analiza deklaracji
- analiza semantyczna instrukcji wraz z generacją L-kodu (kod pośredni)
- generacja wzorców kodu maszynowego
- adresacja zmiennych i budowa struktur danych dla systemu wykonawczego, przetworzenie wzorców kodu maszynowego na tekst w GASSIE, wypisanie wydruku z wplecioną sygnalizacją błędów wykrytych przez kolejne przebiegi.

W pierwszym przebiegu badana jest poprawność syntaktyczna programu; produkowany jest wydruk roboczy zawierający oprócz tekstu źródłowego, uzupełnionego o numerację linii, diagnostykę błędów syntaktycznych. Instrukcje tłumaczone są na kod pośredni na tyle, na ile jest to możliwe w oparciu jedynie o syntaktyczną strukturę programu. Wyrażenia tłumaczone są na odwrotną notację polską, a instrukcje strukturalne zamieniane na symboliczne etykiety i skoki. Produkowany kod wpisywany jest na plik o dostępie bezpośrednim. Równocześnie budowane są struktury danych będące odbiciem deklaracji. Po zakończeniu analizy całego tekstu, w oparciu o utworzone struktury, budowana jest tablica symboli kompilatora, ustalane są powiązania między typami (nazwami klas), badana jest poprawność semantyczna deklaracji i wreszcie jest ustalany porządek modułów uwzględniający zagnieżdżanie i prefiksowanie. Informacje o wykrytych błędach kontekstowych wpisywane są na osobny plik.

Następny przebieg kompilatora przetwarza kod wyprodukowany przez analizator syntaktyczny (ang. parser) na L-kod i bada poprawność semantyczną instrukcji. Instrukcje programu źródłowego są przetwarzane dla każdego modułu osobno, w kolejności odpowiadającej ustalonemu wcześniej porządkowi modułów, przy czym kompilator dwukrotnie przechodzi przez wszystkie moduły. Przy pierwszym przejściu są analizowane instrukcje kodu pośredniego definiujące wartości stałych (np. dla deklaracji **const** n= =2*k+1;) i nie jest przy tym generowany kod. Podczas drugiego przejścia analizowane są instrukcje modułów i produkowany jest na plik sekwencyjny L-kod, mający postać czwórek. Informacje o wykrytych błędach wpisywane są na ten sam plik (w miejsce L-kodu). Podczas generacji L-kodu dokonywane są pewne lokalne optymalizacje, m.in. wyliczanie wszystkich wyrażeń zbudowanych ze stałych. W przebiegu tym zawarta jest cała specyfika LOGLANU.

Kolejny przebieg — generator kodu maszynowego — wczytuje sekwencyjnie L-kod i produkuje, również sekwencyjnie, wzorce kodu maszynowego. Wzorce te są sparametryzowane wartościami stałych, numerami rejestrów i odwołaniami do tablicy symboli w miejscach odpowiadających adresom zmiennych lub adresom modułów.

Dla każdego tłumaczonego modułu ustalana jest liczba niezbędnych słów pamięci na zmienne robocze (wyniki pośrednie). Z właściwości języka wynika, że nie mogą być one trzymane na stosie, lecz muszą być umieszczane w obiekcie (tzn. polu danych). Ze względu na prefiksowanie i przyjętą budowę pól danych dopiero po ustaleniu liczby tych zmiennych roboczych możliwe jest zaadresowanie zmiennych. Jest to zadanie kolejnego przebiegu. Wyznacza on adresy zmiennych i adresy względne w polu danych, buduje i wypisuje dla systemu tzw. prototypy, a więc struktury danych pozwalające na realizację konkatenacji instrukcji, dynamiczne kontrole zgodności typów i parametrów i wreszcie obsługę wyjątków. Następnie wzorce kodu, wyprodukowane przez poprzedni przebieg, są przetwarzane na tekst w asemblerze GASS, z uwzględnieniem już wyliczonych adresów.

Wreszcie, na koniec — wyprowadzany jest wydruk programu. Jeżeli program zawierał błędy, przebieg generujący L-kod powoduje wstrzymanie jego wypisywania. Nie jest wywoływany generator kodu maszynowego ani adresacja, natomiast przed wypisaniem następuje uporządkowanie pliku z sygnalizacją błędów i scalenie z wydrukiem roboczym wraz z dodaniem komentarzy do numerów błędów.

PARAMETRY EKSPLOATACYJNE

Kompilator wymaga co najmniej 20 K słów pamięci i zużywa całą przydzieloną pamięć. Kod kompilatora dla każdego kolejno ładowanego przebiegu zajmuje 13 K słów. Cała nadwyżka, czyli od 7 K, przy minimalnym przydziale, do maksimum 32 K słów jest zużywana na tablicę symboli kompilatora. Rozmiar przydzielonej pamięci wpływa na dopuszczalną liczbę deklaracji w tłumaczonym programie.

Łączna długość wszystkich przebiegów kompilatora wynosi ok. 79 K słów. Kompilator i biblioteka systemu wykonawczego zajmują na dysku ok. dziewięciu cylindrów.

W tabeli podano przykładowe czasy (w sekundach) kompilacji programów różnej długości (w liniach). Tabela uwzględnia czas kompilacji (od tekstu źródłowego w LOGLANIE, po tekst wynikowy w GASSIE), czas przetwarzania przez asembler i wreszcie czas łączenia (ang. linking). Pierwszy z programów to program **block begin end.** Podany czas kompilacji tego programu obrazuje stałą składową czasu kompilacji, a mianowicie czas ładowania przebiegów i inicjacji struktur danych.

Program wynikowy składa się z jądra systemu wykonawczego, tablic stałych (tekstowych i stałych typu **real**, używanych w programie), prototypów (tzn. struktur danych dla systemu wykonawczego), a także ciągu modułów będą-

cych tłumaczeniem modułów w LOGLANIE. Do tego są dołączane — w zależności od programu — procedury wejścia-wyjścia i pomocnicze procedury systemu wykonawczego.

Przykładowe czasy kompilacji, asemblacji i scalania dla programów różnej długości

	Długość programu w liniach				
	1	50	140	720	2603
Czas kompilacji (LOGLAN) [s]	22	25	39	98	323
Czas asemblacji (GASS) [s]	10	10	32	162	315
Czas scalania (EDI) [s]	36	36	55	123	345

Obciążenie programu wynikowego przez bibliotekę systemu wykonawczego, systemu wejścia-wyjścia i procedur standardowych waha się od 4 K do 6 K słów, przy programie korzystającym ze wszystkich możliwości. Dopuszczalna długość wynikowego kodu jest nadal ograniczana do 32 K słów, natomiast dostępna przestrzeń adresowa (do 64 K), używana na wywołania procedur, obiekty klas i tablice dynamiczne jest ograniczona jedynie przez faktycznie dostępną pamięć.

WSTAWKI W ASEMBLERZE

W celu umożliwienia korzystania z procedur bibliotecznych napisanych w innych niż LOGLAN językach programowania, jak również dla umożliwienia oprogramowania niestandardowego wejścia-wyjścia, język i kompilator zostały rozszerzone o wstawki w asemblerze. Pozwalają one na umieszczenie między instrukcjami LOGLANU sekwencji kodu w asemblerze GASS. Wstawki takie mogą być sparametryzowane, przy czym możliwe jest jedynie przesyłanie wartości (obustronne); nie są natomiast dopuszczone parametry proceduralne i typy formalne.

Wstawka w asemblerze jest instrukcją zaczynającą się od słowa **assembler** i kończącą słowem **end** i może wystąpić wszędzie tam, gdzie dopuszczalna jest dowolna instrukcja. Oprócz wstawianego tekstu, specyfikuje ona parametry aktualne wraz ze sposobem ich przekazania.

Parametrami aktualnymi mogą być dowolne wyrażenia (dla parametrów **input**) lub zmienne (dla parametrów **output** i **inout**). Parametry mogą być przekazywane przez rejestry lub przez pamięć. Liczba przekazywanych słów wynika z typu parametru aktualnego. W przypadku przekazywania parametrów przez pamięć, programista ma do dyspozycji 32 kolejne słowa obiektu programu głównego, oznaczane M1—M32. Mogą one być również używane jako zmienne robocze we wstawianym kodzie.

Wstawiany tekst może być dowolnym — byle poprawnym — tekstem w asemblerze GASS (instrukcje, pseudoinstrukcje, komentarze), z ograniczeniami wynikającymi z faktu, że jest kopiowany w środek tekstu modułu. Kompilator nie sprawdza poprawności wstawianego tekstu. Rzecz jasna, wstawiane instrukcje nie mogą zakłócić niezmienników systemu wykonawczego, toteż narzędziem tym należy posługiwać się rozważnie.

Generowany dla wstawki w asemblerze kod składa się z trzech części:
— kodu dla wyliczenia parametrów aktualnych i przekazania wartości parametrów **input** i **inout**
— przepisanego wiernie wstawianego tekstu w GASSIE
— kodu dla pobrania i przekazania wartości parametrów **output** i **inout**.

Przykład wstawki:

```
.... assembler (f(n)*2 : R1, n+1 : M1; output tab(j) : R3);
.....
☆ wstawiany tekst w Gassie
☆ linie od następnej po średniku kończącym nagłówek
  do poprzedzającej end.
☆ (gwiazdka na początku linii oznacza komentarz w Gassie)
.....
end .....
```

Warto podkreślić, że stosowanie wstawek w asemblerze pozwala na tworzenie niemal całego oprogramowania w języku wysokiego poziomu, przy sprowadzeniu części kodowanych w języku symbolicznym do niezbędnego minimum i opakowaniu ich w konstrukcje języka wysokiego poziomu (procedury, funkcje, bloki).

OPERACJE WEJŚCIA-WYJŚCIA

Ze względu na ograniczenia systemu SOM-3, operacje na plikach, zdefiniowane w języku LOGLAN, zostały zrealizowane tylko w wersji sekwencyjnej (istnieje jednak okrojona postać bezpośredniego dostępu).

Program w LOGLANIE dysponuje dwoma standardowymi plikami znakowymi: wejściowym i wyjściowym. Oba te pliki są gotowe do pracy bez konieczności ich programowego otwierania. Z pliku wejściowego można tylko czytać znakowo (**read, readln**) i testować koniec linii (**eoln**) lub koniec pliku (**eof**). W plik wyjściowy można tylko pisać znakowo (**write, writeln**).

Pozostałe pliki używane w programie mają charakter uniwersalny. Można je zapisywać lub odczytywać binarnie (**put, get**) lub znakowo. Pliki te muszą być przed użyciem otwarte — jako lokalne lub związane ze wskazanym strumieniem.

Przy binarnym pisaniu lub czytaniu, możliwe jest przesyłanie obok pojedynczych wartości także całych obiektów, w tym tablic dynamicznych. Przy pisaniu znakowym, liczbę wypisywanych znaków — format można określić dynamicznie przez wyrażenie.

Wprowadzono dodatkową możliwość bezpośredniego dostępu do pliku (realizowaną przez procedury standardowe). Plik jest traktowany jak indeksowana (od 0) tablica rekordów o stałej długości 256 słów. Możliwe jest zapisanie lub odczytanie rekordu o podanym numerze. Rekordy mogą być przesyłane między plikiem a dowolnym obiektem tablicowym typu **array of integer** mającym co najmniej 256 elementów.

PROCEDURY STANDARDOWE

W kompilator została wbudowana biblioteka 38 procedur i funkcji standardowych. „Wbudowanie" należy rozumieć tak, że nie tylko na etapie scalania dołączane są właściwe procedury z biblioteki, ale także — że kompilator „zna" opisy tych procedur i funkcji, a przy wywołaniu bada zgodność parametrów aktualnych z formalnymi i zgodność typu funkcji oraz generuje potrzebne konwersje. Pewne funkcje (np. operacje na ciągach bitów) są ponadto wyliczane przez kompilator, o ile występują ze stałymi argumentami.

Biblioteka obejmuje kilka grup procedur, m.in. funkcje trygonometryczne i numeryczne (pierwiastek, potęgowanie, logarytmy, obcięcia i zaokrąglenia).

Osobną grupę stanowią operacje na ciągach bitów. Argument typu **integer** może być traktowany jako ciąg 16 bitów. Możliwe jest wykonywanie następujących operacji logicznych (bit po bicie) na takich ciągach, jak: negacja, suma i iloczyn logiczny, różnica symetryczna, przesunięcie cykliczne.

Kolejna grupa to procedury i funkcje komunikujące program z otoczeniem. Pozwalają one „pobrać" datę i czas, stan opcji systemu operacyjnego, parametry wywołania programu, stan kluczy na panelu procesora lub mierzyć upływ czasu.

Możliwe jest także przydzielanie strumieni do urządzeń fizycznych, pomijanie zbiorów w przód lub w tył, przewijanie strumienia, dopisywanie znacznika końca zbioru, czytanie lub pisanie z bezpośrednim dostępem.

Biblioteka zawiera także generator liczb losowych, z możliwością ustalenia początku serii.

OPCJE KOMPILACJI

Za pomocą opcji, czyli binarnych przełączników, programista może wpływać na działanie kompilatora. Opcje pozwalają na włączenie lub wyłączenie wydruku, śledzenie programu, wyłączanie dynamicznych kontroli (test na istnienie obiektu, badanie poprawności indeksu i zgodności typów) oraz pozwalają wpływać na dokonywane optymalizacje.

Opcje mogą mieć działanie lokalne lub globalne i być ustawiane — odpowiednio — z wewnątrz lub z zewnątrz

tekstu programu. Opcje mają jednoliterowe nazwy, po których występuje plus albo minus. Wewnętrzne ustawianie opcji odbywa się analogicznie jak w kompilatorach PASCALA. Wystąpienie w tekście programu specjalnej postaci komentarza oznacza ustawienie opcji, mające skutek od początku najbliższej instrukcji — aż do kolejnej zmiany tych opcji (lub do końca programu). Na przykład — (☆ $L-M--T+☆) oznacza wyłączenie wydruku (L-) i kontroli dostępu do obiektów (M—) oraz włączenie dynamicznej kontroli zgodności typów (T+). Takie rozwiązanie pozwala na wypisywanie tylko fragmentów tekstu (ważne przy wielokrotnym kompilowaniu długich programów) lub wyłączanie dynamicznych kontroli w tych procedurach, których poprawność została zweryfikowana.

Możliwe jest także zewnętrzne (względem tekstu programu) wymuszanie wartości opcji, przez podanie ciągu opcji przy wywołaniu kompilatora. W tym przypadku dla każdej podanej zewnętrznie opcji podana wartość obowiązuje w całym tekście programu, a jej wewnętrzne zmiany są ignorowane. Dzięki temu można przeforsować wybrane wartości opcji bez konieczności wyszukania wszystkich zmieniających je miejsc w programie. Rzecz jasna, dla opcji nie ustawionych w żaden z powyższych sposobów, obowiązują ustalone wartości domyślne.

O ile możliwość wyłączania wydruku nie wymaga komentarzy, to pozostałe opcje wymagają pewnych wyjaśnień.

W przypadku poprawnego logicznie programu warto skorzystać z możliwości wyłączenia dynamicznych kontroli, ponieważ przyspiesza to działanie programu wynikowego i skraca kod wynikowy.

Przykładowo — wyłączenie kontroli indeksów i dostępu do obiektów przyspieszyło algorytm mnożenia macierzy (używający dwuwymiarowych tablic dynamicznych) o 35% (przy mnożeniu macierzy typu real o wymiarach 60☆100 razy 100☆60 czas zmniejszył się z 166 s do 109 s, a przy wymiarze 40☆100 — z 81 do 50 s). W innym natomiast, wyrafinowanym programie intensywnie używającym typów i funkcji formalnych (generującym kolejne liczby pierwsze przy użyciu kolejki priorytetowej zrealizowanej jako abstrakcyjna struktura danych, sparametryzowana typem elementu i relacją porządku), wyłączenie dynamicznej kontroli typów przyspieszyło działanie programu o 20% i skróciło kod (bez uwzględnienia systemu wykonawczego) o 16%.

Jedna z opcji umożliwia także prymitywne śledzenie wykonywanego programu (najnowsza wersja kompilatora jest już rozbudowana o wyrafinowany program wspomagający uruchamianie — (ang. debugger). Dla fragmentów programu, dla których opcja ta była włączona, kompilator generuje na początku każdej instrukcji odwołanie do procedury systemu wykonawczego z parametrem określającym numer linii w tekście źródłowym. W niewielkim stopniu wydłuża to kod wynikowy i czas wykonania programu, pozwala natomiast w przypadku wystąpienia błędu, podczas wykonania programu, określić odpowiedzialną za błąd linię tekstu źródłowego. Ponadto, o ile była włączona odpowiednia opcja podczas wykonywania programu, wypisywane są numery linii odpowiadające kolejno wykonywanym instrukcjom.

DIAGNOSTYKA BŁĘDÓW

Autorzy kompilatora pragnęli uczynić diagnostykę błędów możliwie czytelną dla programisty, a przy okazji uniknąć pewnych niedogodności znanych z innych kompilatorów. Przyjęte rozwiązania są pewnym kompromisem między wygodą a kosztami.

Generalnie rzecz biorąc — informacje o błędzie (numer błędu i krótki opis) są umieszczane pod błędną linią. Nie trzeba zatem co chwila zerkać na koniec tekstu, by odczytać wypisane komentarze do numerów błędów.

Ze względu na wieloprzebiegową analizę tekstu programu (rozdzielenie analizy syntaktycznej, analizy deklaracji i analizy instrukcji) informacje o błędach syntaktycznych zrealizowano inaczej niż o pozostałych rodzajach błędów. Błędy syntaktyczne są lokalizowane z dokładnością do jednego znaku i wskazywane przez wykrzyknik umieszczony pod miejscem błędu. Obok podany jest numer błędu, a poniżej komentarz. Czasem korekcja błędu dokonywana przez analizator syntaktyczny nie jest najlepsza i powoduje jeszcze kilkakrotną jego sygnalizację.

Bardziej skomplikowanie niż dla błędów syntaktycznych wygląda diagnostyka błędów semantycznych. Zarówno analiza deklaracji, jak i instrukcji jest dokonywana po zakończeniu analizy syntaktycznej i przebiega w kolejności całkowicie różnej od tekstowej. Informacje o wykrytych błędach wpisywane są na plik, po czym — po zakończeniu kompilacji — plik ten jest sortowany względem numerów linii oraz łączony z plikiem wyprodukowanym przez analizator syntaktyczny, a zawierającym tekst programu z zaznaczonymi błędami syntaktycznymi.

Analiza deklaracji dokonywana jest jedynie na podstawie tablicy symboli, w oderwaniu od tekstu źródłowego. Precyzyjne umiejscowienie błędu w tekście wymagałoby istotnego rozbudowania tablicy symboli, na co nie pozwalały ograniczenia pamięciowe. Podobnie, dla precyzyjnego umiejscowienia błędów semantycznych w instrukcjach, trzeba by było znacznie wydłużyć kod pośredni, produkowany przez analizator syntaktyczny, dołączając do każdego symbolu informację o jego położeniu w tekście źródłowym (numer linii i pozycja w linii). Ostatecznie informacja o błędzie podaje — poza numerem błędu i odpowiednim komentarzem — identyfikator, którego błąd dotyczy (o ile to ma sens). Informacja ta umieszczana jest pod błędną linią. Nieznane pozostaje jedynie położenie błędu w linii. Na ogół jest to informacja wystarczająca, pewne wątpliwości co do miejsca wystąpienia błędu mogą powstać jedynie w przypadku, gdy linia zawiera kilka instrukcji (deklaracji) lub gdy instrukcja deklaracja) rozciąga się na kilka linii — diagnostyka wypisywana jest wówczas pod pierwszą z nich, mimo że błąd mógł wystąpić dalej. Są to jednak sytuacje rzadkie i zazwyczaj podany identyfikator pomaga w zlokalizowaniu błędu.

Wydaje się, że dobrze można ocenić korekcję błędów semantycznych. Pojawienie się np. niezadeklarowanego identyfikatora lub identyfikatora nielegalnego w danym kontekście jest sygnalizowane jednokrotnie i nie powoduje lawiny błędów sygnalizowanych przy każdym kolejnym jego wystąpieniu lub każdej dalszej niezgodności.

Informacje o błędach wypisywane są zawsze, również w przypadku wyłączenia wydruku. Wypisywane są wtedy błędne linie wraz z otoczeniem, co ułatwia wykrycie błędu. Jeśli zważyć, że przy poprawianiu długich programów często praktykowane jest wyłączenie (całkowite lub częściowe) wydruku, to łatwo docenić wygodę płynącą z wypisania błędnych linii wraz z diagnostyką, zamiast jedynie komunikatu „BYLY BLADY" (oryginalna diagnostyka kompilatora FORTRANU na MERZE).

BŁĘDY CZASU WYKONANIA

System wykonawczy LOGLANU gwarantuje pełne bezpieczeństwo działania programu tzn. nie może się zdarzyć, że program nawet niepoprawny pójdzie w maliny (dla osób nie oswojonych z żargonem: zacznie zachowywać się w losowy sposób). Wynika to z faktu, że kontrolowane są wszystkie dostępy przez kropkę do atrybutów obiektów (sprawdzenie, czy obiekt nie został usunięty), wartości indeksów przy dostępie do tablicy, zgodności typów, parametrów itd.

Oczywiście, wyłączenie dynamicznych kontroli lub też zakłócenie niezmienników systemu wykonawczego poprzez nieprzemyślaną wstawkę w asemblerze może zakłócić to bezpieczeństwo. Warto podkreślić, że chociaż pełna dynamiczna kontrola powoduje pewien wzrost kosztów (długość kodu wynikowego, czas wykonania), to jednak znacznie przyspiesza uruchamianie programu.

Oprócz błędów wykrywanych przez testy umieszczone w kodzie wynikowym lub przez procedury systemu wykonawczego, „przechwytywane" są także błędy wykryte przez sprzęt (przerwania) lub system operacyjny. W każdym przypadku wypisywany jest komunikat podający rodzaj błędu i numer linii w tekście źródłowym, odpowiadający instrukcji odpowiedzialnej za błąd — o ile oczywiście podczas kompilacji nie wyłączono odpowiedniej opcji.

Programista przewidując błędy, może wykorzystać mechanizm obsługi wyjątków i instrukcji lastwill dla własnej obsługi błędów i kontynuacji obliczeń.

OBSŁUGA WYJĄTKÓW

Mechanizm obsługi wyjątków (ang. exception handling) pozwala na inny styl programowania, co wymagałoby odrębnego omawiania — ale także może być bardzo pomocny przy uruchamianiu programu.

W języku LOGLAN wyróżnia się dwa rodzaje wyjątków: pojawienie się zadeklarowanego w programie (i być może sparametryzowanego) sygnału oraz wystąpienie błędu (sygnały predefiniowane). Zatrzymajmy się przy tych ostatnich. W opisywanym kompilatorze wyróżniono następujące sygnały systemowe:

ACCERROR — próba dostępu do nie istniejącego obiektu (referencja = note)

CONERROR — wartość indeksu wykraczająca poza zakres tablicy

LOGERROR — wszystkie błędy związane z przekazywaniem sterowania

MEMERROR — przepełnienie pamięci

NUMERROR — błędy związane z działaniami arytmetycznymi: dzielenie przez zero, nadmiar lub niedomiar zmiennopozycyjny

TYPERROR — niezgodność typów w instrukcji przypisania lub przy transmisji parametrów

SYSERROR — błędy związane z komunikacją z systemem operacyjnym SOM-3, w tym błędy w instrukcjach wejścia-wyjścia.

Programista może w swoim programie przewidzieć wystąpienie konkretnego błędu i zdefiniować jego obsługę (ang. **handler**), może też zdefiniować ją w sposób uniwersalny (**others**) — dla obsłużenia dowolnego błędu. Może także — używając instrukcji **lastwill** — określić akcję do wykonania przy nienormalnym zakończeniu pracy modułu (np. wypisać stosowny komunikat i wartości interesujących zmiennych).

Co się będzie działo po wykryciu błędu w wykonywanym programie? Najpierw zostanie przeszukany dynamiczny łańcuch obiektów (nieco upraszczając: ciąg wywołujących się procedur), poczynając od ostatnio wykonywanego, aż do znalezienia odpowiedniej obsługi błędu. Następnie wykonywane są instrukcje tej obsługi aż do instrukcji **terminate** lub **wind**, po czym zostanie „zwinięty" łańcuch poniżej obiektu zawierającego obsługę i obliczenia będą kontynuowane do obiektu zawierającego obsługę (przy **wind**) lub obiektu go wywołującego (przy **terminate**). Warto dodać, że w przypadku deklarowanych sygnałów możliwe jest także kontynuowanie obliczeń bez zwijania łańcucha obiektów. „Zwijanie" łańcucha polega na wykonaniu instrukcji **lastwill** z kolejnych obiektów łańcucha. W razie nieznalezienia obsługi zwijany jest cały łańcuch i program — po wykonaniu odpowiednich instrukcji **lastwill** — kończy swoje działanie. Wypisywany jest wówczas stosowny komunikat informujący o błędzie.

* * *

Język programowania LOGLAN jest używany od początku 1982 roku w kilkunastu ośrodkach. W odróżnieniu od pierwszej, niedostatecznie przetestowanej i niefortunnie udostępnionej kilku użytkownikom wersji kompilatora LOGLANU 78, obecna wersja kompilatora LOGLANU 82 została dość gruntownie sprawdzona — była ona przez półtora roku eksploatowana przez studentów Instytutu Informatyki UW, nie licząc innych ośrodków.

Zespół autorów LOGLANU (języka i implementacji) jest przekonany, że z obecnym kompilatorem język LOGLAN 82 stanowi wartościowe narzędzie pracy dla użytkowników MERY 400. Dla rozszerzenia kręgu użytkowników LOGLANU, czynione są przygotowania do zainstalowania tego języka na maszynach JS i SM, w oparciu o istniejącą implementację.

LITERATURA

[1] Findeisen P., Gburzyński P.: GASS — MERA 400 Assembly Language Reference Manual. Sprawozdania Instytutu Informatyki Uniwersytetu Warszawskiego nr 117 i 118

[2] International Summer School of the Programming Language LOGLAN, Zaborów, 1983

[3] Kompilator języka programowania Loglan 82 na Merę 400. Podręcznik użytkownika

[4] Kreczmar A., Salwicki A.: Język programowania LOGLAN. Informatyka nr 7, 8—9/1982 i 1/1983

[5] Report on the Loglan 82 programming language. PWN, Warszawa-Łódź, 1984.

KONRAD JABŁOŃSKI
ŚCO Cyfronet
Instytut Energii Atomowej
Świerk

Język programowania BCPL (2)

W pierwszej części artykułu scharakteryzowałem podstawowe właściwości języka. Poniżej omawiam przykładowy tekst programu oraz zasady implementacji.

STANDARDOWA BIBLIOTEKA

Programy pisane w języku BCPL muszą korzystać ze standardowej biblioteki zawierającej procedury i funkcje umożliwiające realizację operacji wejścia-wyjścia. Komunikacja z tą biblioteką odbywa się za pomocą zmiennych globalnych, przy czym tablica odpowiednich wartości globalnych adresów jest zwykle dostępna razem z kompilatorem (stanowi jego uzupełnienie). Z kolei system operacyjny komputera docelowego, wykonującego program przygotowany w języku BCPL, musi zawierać pakiet biblioteczny standardowych procedur i funkcji. System operacyjny musi mieć także specjalny podprogram organizujący maszynę wirtualną, na której będzie działać program użytkowy. Podprogram ten zwykle organizuje stos maszyny wirtualnej i porządkuje pewne wskaźniki niezbędne do jej działania, choć ich znajomość nie jest konieczna do pisania i uruchamiania programów.

Część standardowej biblioteki musi być napisana w języku asemblera komputera docelowego i tworzy się ją w ramach implementacji języka — jako rozszerzenie istniejącego systemu operacyjnego. Pozostała część jest napisana w języku BCPL i może być dołączona każdorazowo do programu w fazie kompilacji, bądź dołączana do systemu operacyjnego komputera docelowego. W tym drugim przypadku korzysta się z niej analogicznie jak z części napisanej w języku asemblera.

Podział na dwie grupy wynika z faktu, że pewne funkcje lub procedury zależą od sprzętowych rozwiązań konkretnego komputera i nie mogą być uniwersalne. Procedury i funkcje standardowe, które mogą być zapisane w sposób niezależny od komputera, na jakim będą wykonywane, pisze się w języku BCPL.

Poniżej omawiam niektóre procedury i funkcje standardowe. Do najważniejszych należą SELECTINPUT i SELECTOUTPUT. Służą one do przełączania strumieni wejściowych i wyjściowych w zależności od potrzeb programu, przy czym ich argumenty mogą być liczbowe lub symboliczne, zależnie od implementacji. Uzupełnieniem wymienionych procedur są funkcje FINDINPUT i FINDOUTPUT, wywoływane wraz z argumentami określającymi nazwy urządzeń i zbiorów, co umożliwia dynamiczne dopasowywanie się do sytuacji. Podstawową operację wejścia realizuje funkcja wczytania znaku RDCH, a podstawową operację wyjścia — procedura WRCH. Strumienie wejścia i wyjścia zamyka się procedurami ENDREAD i ENDWRITE.

Przy użyciu powyższych procedur i funkcji tworzy się zestaw funkcji i procedur wyższego poziomu, jak procedury wprowadzania liczb, np. WRITED, WRITEN, WRITEOCT, WRITEHEX (służące do wyprowadzania liczb w zapisie dziesiętnym w narzuconym formacie, bez formatu, w zapisie ósemkowym i szesnastkowym). Należą do nich także procedury WRITES, UNPACKSTRING i PACKSTRING, przeznaczone do wyprowadzania ciągu znaków oraz jego rozpakowania i upakowania.

Osobną i ważną grupę stanowią: procedura LONGJUMP i funkcja LEVEL. Procedura LONGJUMP z odpowiednimi argumentami pozwala wykonać skok do dowolnego miejsca w programie z uwzględnieniem stopnia zagnieżdżenia, z czym wiąże się redukcja stosu maszyny wirtualnej. Właściwe miejsce w programie markuje się funkcją LEVEL, która zapamiętuje przy użyciu specjalnych zmiennych programu zarówno wartość wskaźnika stosu wirtualnego, jak i adres etykiety wskazanej na przewidywane miejsce skoku.

PRZYKŁADOWY TEKST PROGRAMU

Poniżej przedstawiono przykład zapisu programu w języku BCPL. Jest to deklaracja procedury bibliotecznej drukowania znaku — WRITEC. Procedura ma jeden parametr, którym jest znak alfanumeryczny, oznaczony symbolicznie przez CH.

Na początku programu zadeklarowano dwie zmienne globalne: OUTPUT i PACKSTRING. Zakłada się, że pod adresem względnym 4 w tablicy GLOBAL w chwili wykonania programu będzie umieszczony prawidłowy adres wektora. Podobnie zakłada się, że pod adresem względnym 20 w odpowiedniej chwili znajdzie się adres specjalnej procedury systemowej (napisanej w języku asemblera komputera docelowego), mającej na celu rozpakowanie ciągu znakow upakowanych w sposób charakterystyczny dla danego komputera. Znajomość adresów względnych w obszarze globalnym jest konieczna do prawidłowego programowania. Część tych adresów określa system wykonawczy (ang. runtime system).

W tablicy MANIFEST zdefiniowano nazwy odpowiadające stałym 0, 1, 2, 3, 120, używanym w programie. Wartości tych nazw są wykorzystywane jedynie w czasie kompilacji programu.

W tablicy STATIC zdefiniowano tylko jedną zmienną statyczną COUNTER, pełniącą rolę licznika wołań procedury. Zmienna COUNTER jest dostępna dla procedury w celu zaliczania jedynek, natomiast poza procedurą jest dostępna także dla innych części programu, np. dla innych procedur w celu kontroli.

Procedura WRITEC po wywołaniu tworzy dwie zmienne dynamiczne (na stosie). Zmienna LINE ma wartość początkową utworzoną z wartości pierwszego elementu wektora OUTPUT, a zmienna NEXT ma wartość początkową utworzoną z wartości drugiego elementu wektora OUTPUT zwiększonego o 1.

Następnie procedura bada swój parametr wejściowy, czyli CH. Jeśli parametr ten jest równy liczbie 10 (dziesiętnie), oznacza to, że jest to kod ASCII znaku zmiany linii. Wyrażenie '☆N' oznacza w języku BCPL liczbę 10. W takim przypadku wartością zerowego elementu wektora LINE staje się wartość drugiego elementu wektora OUTPUT. Zatem zawartość pierwszego elementu wektora OUTPUT powinna być adresem innego wektora.

Z kolei, zostaje wywołana procedura systemowa PACKSTRING o dwu identycznych parametrach, którymi są adresy buforów przechowujących rozpakowane i upakowane znaki. W tym przypadku jest to zresztą ten sam bufor.

Po wykonaniu operacji wyprowadzenia ciągu znaków z bufora następuje wyzerowanie wartości drugiego elementu wektora OUTPUT, gdyż zawartość bufora została skonsumowana. Należy zwrócić uwagę, że operacja fizyczna jest realizowana przez specjalną procedurę (pisaną zwykle w języku asemblera), której adres znajduje się w trzecim elemencie wektora OUTPUT, nazwanym OUTPUT%TRANSFER. Procedura ta wymaga podania dwu parametrów, którymi są zawartość zerowego elementu wektora OUTPUT oraz adres bufora ze znakami LINE. Zawartość zmiennej statycznej COUNTER zostaje zwiększona o 1 i następuje wydrukowanie kolejnej linii tekstu.

Jeśli jednak parametr CH nie jest znakiem zmiany linii to następuje sprawdzenie, czy zmienna NEXT nie jest większa od 120. W procedurze przyjęto, że 120 znaków alfanumerycznych może znajdować się w linii tekstu. Jeśli zgromadzony tekst jest krótszy niż maksymalnie dopuszczalny, to w wektorze OUTPUT umieszczona zostaje wartość o 1 większa (zaliczenie znaku), a w wektorze LINE (w buforze) — aktualny znak. W tym przypadku nie inicjuje się operacji wyprowadzania.

Jeżeli kolejny znak nie jest znakiem zmiany linii, ale bufor gromadzący znaki może zostać przepełniony, wtedy najpierw zostaje wywołana ta sama procedura WRITEC, lecz z parametrem będącym znakiem zmiany linii, co oczywiście wywoła wyprowadzenie wszystkich znaków alfanumerycznych zgromadzonych do tej pory. Zmienna NEXT przyjmuje wskutek tego wartość 1, gdyż znak niesiony przy pierwszym wywołaniu procedury jeszcze czeka. Dalsze postępowanie jest identyczne jak w poprzednim przypadku, tj. czekający znak zostanie umieszczony w buforze LINE jako pierwszy.

Interesujący w tym przykładzie jest fakt rekurencyjności wywołania procedury, co jest typowe dla języka BCPL. Właściwość tę uzyskano dzięki pracy ze stosem, przy czym jest to stos specjalnie tworzony przez system wykonawczy na użytek programu pisanego w BCPL. Bez względu na to, czy komputer rzeczywisty ma stos lub pracuje ze stosem sprzętowym bądź systemowym, wymieniony stos jest tworzony zawsze niezależnie.

STOSOWANIE JĘZYKA

Tekst źródłowy użytkowego programu napisanego w języku BCPL podlega kompilacji za pomocą kompilatora języka BCPL, wskutek której uzyskuje się program wynikowy w języku OCODE. Postać ta może podlegać dalszej translacji na zapis w języku symbolicznym (języku asemblera) dla komputera określonego typu. Dokonuje się tego przy użyciu przeznaczonego dla tegoż komputera translatora kodu OCODE. Uzyskany po translacji tekst w języku symbolicznym podlega kolejnej translacji przy użyciu asemblera komputera określonego typu, wskutek czego otrzymuje się bądź wynikowy kod binarny, nadający się do bezpośredniego wprowadzenia do pamięci komputera, bądź kod pośredni służący do wygenerowania kodu binarnego komputera.

Wielostopniowa generacja kodu binarnego na podstawie tekstu w języku BCPL, choć pozornie skomplikowana, pozwala uzyskać takie właściwości, jakich nie mają implementacje wielu współcześnie tworzonych języków maszynowych. Przede wszystkim wykorzystuje się przy tym oprogramowanie podstawowe, które ma każdy komputer.

Wprowadzenie pośredniej postaci zapisu w OCODE pozwala łatwo rozpowszechniać programy opracowane w języku BCPL. Program sprawdzony nie tylko pod względem formalnym, ale i merytorycznym można uruchomić na dowolnym komputerze — pod warunkiem, że na tym komputerze jest dostępny translator OCODE i biblioteka podstawowa, z której korzystają napisane programy. Opracowanie translatora OCODE i biblioteki BCPL jest o wiele prostsze niż pisanie kompilatora BCPL dla konkretnego komputera.

Szczególnie korzystne jest istnienie jednego większego komputera z bogatym oprogramowaniem, biblioteką i kompilatorem BCPL-u oraz — grupy różnych mini- i mikrokomputerów posiadających tylko translatory OCODE i własne asemblery. W takiej sytuacji opracowywanie oprogramowania dla całej rodziny mini- i mikrokomputerów może odbywać się wydajnie na dużym komputerze, bez konieczności tworzenia symulatorów poszczególnych mini- i mikrokomputerów, które zresztą nigdy nie są dostatecznie wierne.

IMPLEMENTACJA

Implementacja języka może zostać wykonana dwoma sposobami. Jeżeli istnieje kompilator języka BCPL na jednym komputerze i może wytwarzać kod OCODE z dostarczonego tekstu programu, to wtedy opracowuje się, najczęściej w języku symbolicznym docelowego komputera, translator TRAN1 kodu wejściowego (OCODE) na kod wyjściowy (asemblerowy). Dodatkowo wyposaża się system operacyjny komputera docelowego w niezbędną bibliotekę BCPL i implementacja jest zakończona. Programy przygotowuje się na jednej maszynie (gdzie jest kompilator), a wykonuje się

dokończenie na str. 21

Poniższy artykuł dedykujemy szczególnemu kręgowi odbiorców: „profesjonalnym hobbystom". Tym razem gratka dla tych, którzy mozolnie próbują unowocześnić nasz przemysł wprowadzając mikroprocesorowe sterowniki.

Bezspornie najbardziej efektywną obsługę sterowników mikroprocesorowych zapewnia oprogramowanie pisane w języku ASSEMBLER. Trzeba jednak uwzględnić fakt, że współpraca specjalistów różnych dziedzin z programistami przebiega dość opornie. Optymalnym wyjściem byłoby tworzenie oprogramowania przez tych, którzy najlepiej rozumieją istotę procesów kontrolowanych przez komputer. Taką szansę stwarza jeden z najłatwiejszych do opanowania języków — BASIC. Informacja o dostępnej wersji tego języka, umożliwiającej sterowanie w czasie rzeczywistym (i to pracującej pod system CP/M!) może okazać się przełomowa dla całej gamy nowych zastosowań mikroprocesorów do sterowania.

INDUSTRIAL REAL-TIME BASIC – dla mikrokomputerów

INDUSTRIAL REAL-TIME BASIC (IRTB) został zdefiniowany jako moduł stanowiący rozszerzenie klasycznego języka Basic. IRTB jest przeznaczony do wykorzystania w sterowaniu automatyzacją procesów technologicznych i w monitorowaniu otoczenia komputera, czyli w zastosowaniach, które można rozdzielić na określoną liczbę współbieżnych procesów — w dużym stopniu niezależnych i asynchronicznych, ale okazjonalnie się z sobą komunikujących i synchronizujących. Program dla takich zastosowań jest więc z natury rzeczy „wielowątkowy" i musi mieć na ogół możliwość działania w nieograniczonym horyzoncie czasowym.

Język BASIC był niedawno prezentowany na łamach INFORMATYKI [1] i w związku z tym niniejszy artykuł koncentrować się będzie na zagadnieniach związanych z implementacją IRTB.

IRTB zawdzięcza swoją atrakcyjność prostocie koncepcyjnej zastosowanych mechanizmów oraz... brakowi liczącej się konkurencji — oczywiście w klasie prostych zastosowań przemysłowych i laboratoryjnych. Pewną rolę grać też będzie niewątpliwie przyzwyczajenie użytkowników (szczególnie nie będących informatykami), którzy dotychczas skazani byli na programowanie w językach asemblerowych lub stosowali jedną z wielu wersji języka BASIC, adaptowanych do celów sterowania [1]). Obecnie odmiany IRTB stanowią podstawowe oprogramowanie kilku niedawno wprowadzonych na rynki zachodnie uniwersalnych sterowników mikrokomputerowych. Do dziś nie powstała jednak na świecie pełna (zgodna z normą) implementacja IRTB.

Zalety IRTB w oprogramowywaniu układów sterowania były inspiracją podjęcia w PPZ COMPUTEX pracy nad własną implementacją tego języka. Pilotująca implementacja została dokonana na systemie CS-80 i nosi nazwę INDUSTRIAL BASIC.

Przewidywany nakład pracy oraz zapewnienie możliwości przenoszenia na inne systemy były decydującymi czynnikami w wyborze sposobu implementacji. Zdecydowano się na interpreter, mimo że powoduje to znaczne spowolnienie wykonywania programów użytkowych. W systemie z założenia pracującym w czasie rzeczywistym może się to okazać istotnym ograniczeniem. W zrealizowanej implementacji gra ono rolę dopiero przy potrzebie reakcji w czasie krótszym od 0,1—0,2 s.

W chwili rozpoczęcia prac opierano się na propozycji standardu IRTB z września 1981 [2]. Późniejsze poprawki (do rozdz. 5.2 i 5.6 tego dokumentu) nie miały wpływu na przygotowywany program interpretera, ponieważ dotyczyły nieimplementowanych elementów języka: deklaracji struktur danych oraz instrukcji niedeterministycznego oczekiwania zadania na komunikację z innym zadaniem (blok SELECT). Ponadto, syntaktyka języka została w kilku miejscach w niewielkim stopniu zmieniona, co podyktowane było względami implementacji. Język rozbudowano też o instrukcje spoza standardu, przeznaczone głównie do współpracy z zasobami mikrokomputera (CLOCK, PEEK, POKE, wywołanie podprogramu w kodzie maszynowym).

Program INDUSTRIAL BASIC może być wykorzystywany w dwóch wersjach:

— uruchomieniowej, jako program biblioteczny dyskowego systemu operacyjnego MAGOS (kompatybilnego z CP/M)

— rezydencyjnej, umieszczanej w pamięci EPROM sterownika mikrokomputerowego.

Pierwsza wersja stosowana jest na etapie projektowania i testowania programów w warunkach laboratoryjnych,

[1]) W Polsce był to przede wszystkim implementowany przeze mnie wielozadaniowy BASIC RT, oferowany w różnych wariantach przez ISEP PW, PPZ COMPUTEX, a ostatnio także MIKRONIKĘ z Poznania.

Na Węgrzech można...

Dla wielu rodaków Węgry to kraj taniego wina i sweterków, za którymi uganiają się polskie turystki. Jest to jednak również kraj, w którym zaskakująco wiele (a realia w końcu są niemal identyczne) dzieje się w mikroinformatyce. Zacznijmy od rzeczy nam najbliższej: w nakładzie 30 tys. egzemplarzy ukazuje się tam dwumiesięcznik „Mikro" — μ-MIKROSZAMITOGEP-MAGAZIN (biorąc pod uwagę liczbę ludności, odpowiadało by to u nas 100 tys. nakładu). Pismo nie tylko dobrze się sprzedaje, ale również przynosi dochód — pokaźna jego część pochodzi z ogłoszeń. Okazuje się bowiem, że na Węgrzech opłaca się np. reklamować... COMMODORE C64. Wprawdzie kompletny system (komputer, monitor, napęd dysków elastycznych) kosztuje ok. 70 tys. forintów (prawie trzyletnia przeciętna płaca), ale i nasza cena wolnorynkowa za kompletny system kształtuje się w takich granicach. Tyle że na Węgrzech można C64 kupić w sklepie. Za dwumiesięczne wynagrodzenie (10 tys. forintów) można natomiast kupić prosty mikrokomputer węgierskiej produkcji. Dokładnie odwrotnie niż u nas (cena MERITUM odpowiada obecnie wolnorynkowej cenie C64). Wracając jeszcze do ogłoszeń: zamieszczają je również producenci sprzętu profesjonalnego — widocznie nie mogą liczyć (odwrotnie niż nasi), że cokolwiek zrobią to i tak rozejdzie się spod lady.

Na tym jednak nie koniec rewelacji z węgierskiego rynku. Otóż na Węgrzech można także nabyć zachodnie profesjonalne mikrokomputery. W tym nie tylko IBM PC ale nawet... MACINTOSHA. Teraz to już jasne jakim cudem Węgrzy oferowali na Zachodzie oprogramowanie graficzne do tego mikrokomputera.

Mikrokomputerowa oferta na węgierskim rynku tylko częściowo finansowana jest za „państwowe" dolary. Tak jak u nas niektórzy dorabiają na sprowadzaniu z Zachodu końcówek do długopisów — na Węgrzech można dorobić... sprowadzając mikrokomputer. Cała operacja jest zupełnie legalna i co najważniejsze — istnieją precyzyjne przepisy określające opodatkowanie (ustalona jest jednoznaczna kwota cła dla komputerów przeznaczonych do odsprzedania — i na tym koniec). W ten prosty sposób państwo nie tylko nie dokłada do upowszechniania mikroinformatyki, lecz wprost przeciwnie — ciągnie z tego określone zyski. Na pośrednictwie korzysta również nabywca, który dostaje komputer przetestowany i... z gwarancją. Czy to możliwe abyśmy mogli w naszym kraju zastosować równie prostą metodę?

Wracając jeszcze do piśmiennictwa mikroinformatycznego na Węgrzech. Przemiennie z dwumiesięcznikiem „Mikro" mają ukazywać się monotematyczne broszury (tu kolejna różnica: mikroKLAN próbuje wydawać broszury — natomiast Węgrzy już je drukują). Przygotowywana jest broszura opisująca sposób samodzielnego złożenia mikrokomputera, inna z kolei poświęcona zostanie językowi LOGO. Zestawiając tematy okazuje się, że wiele pomysłów jest niemal identycznych. Jednak — z relacji węgierskich kolegów — ich droga od pomysłu do realizacji, choć również nie jest usłana różami, wygląda na znacznie krótszą. I jeszcze jedno: honorarium, jakie można otrzymać na Węgrzech za dobry tekst, zbliżone jest do miesięcznej pensji inżyniera. Takie podejście daje możliwość „pozyskania" naprawdę dobrych autorów. W konsekwencji, mimo umowy z zachodnioniemieckim pismem CHIP, umożliwiającej przedruk dowolnych publikowanych tam materiałów, Węgrzy jeszcze ani razu nie potrzebowali z tej możliwości korzystać.

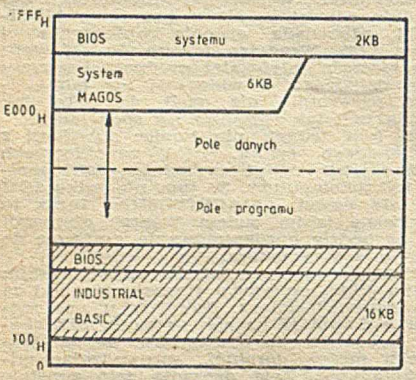
Do tej pory zwolennicy mikroinformatyki z zazdrością patrzyli na kraje zachodnie. Okazuje się, że możemy również zazdrościć krajom o tym samym systemie. Tylko na czym to polega, że tam można, a u nas niezupełnie?

AJP

docelowo eksploatowanych już na rzeczywistym obiekcie przez wersję drugą. Takie podejście skraca czas przygotowywania programów.

INDUSTRIAL BASIC składa się z następujących modułów:
— edytor programów użytkowych (tylko w wersji uruchomieniowej)
— szybki interpreter
— pakiet arytmetyki zmiennoprzecinkowej
— biblioteka funkcji matematycznych (opcja)
— system operacyjny czasu rzeczywistego
— moduł inicjacji
— moduł BIOS.

Moduł BIOS (ang. Basic Input Output System) umożliwia szybkie przystosowanie programu do praktycznie dowolnego mikrokomputera wykorzystującego procesor Z80. Umieszczona na początku modułu tablica skoków powoduje, że przeniesienie programu na inny sprzęt wymaga jedynie napisania nowych procedur współpracy z nowymi zasobami sprzętowymi.



Rys. 1. Mapa pamięci komputera

Użytkownik ma do swojej dyspozycji — dla programu w języku BASIC i jego dynamicznych struktur danych — do 45 KB pamięci. Podział pamięci na pole programu i pole danych jest płynny, jednak program użytkowy nie może nałożyć się na system operacyjny wymagany dla odczytu-zapisu programu na dyskietce.

Program użytkownika składa się z dwóch części: segmentu deklaracji globalnych i segmentu zadań współbieżnych. Deklaracje wprowadzają do programu nazwy obiektów globalnych:
— „wiadomości" wymienianych między zadaniami o treści numerycznej i określonej długości (deklaracja MESSAGE)
— sygnałów generowanych programowo lub przerwaniem zewnętrznym (deklaracja PROCESS EVENT)
— portów we-wy cyfrowego i analogowego (deklaracja PROCESS INPUT/OUTPUT)
— zmiennych wspólnych dla wszystkich zadań (deklaracja SHARED).

Segment zadań współbieżnych obejmuje zbiór sekwencyjnie definiowanych programów (zadań), ujętych w nawiasy syntaktyczne PARACT (ang. PARallel ACTivity) i END. Wielkość pola danych, koniecznego dla zadania, może być za każdym razem dowolnie deklarowana na podstawie oszacowania dokonanego przez programistę.

Z chwilą podania komendy RUN następuje wejście w moduł inicjacji, który między innymi przydziela bufory związane z zadeklarowanymi obiektami globalnymi. Sprawdzana jest też poprawność budowy programu. Pomyślne zakończenie tych operacji sygnalizowane jest komunikatem:

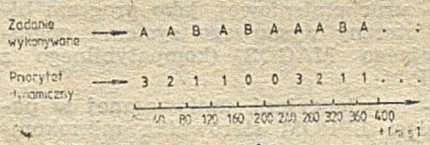
Preprocessing completed, time xx:xx:xx

W tym momencie rozpoczyna swój bieg zadanie współbieżne, które umieszczone jest jako syntaktycznie pierwsze w programie. Wszystkie inne zadania — zgodnie z wymaganiem standardu — muszą być uruchomione jawnie za pomocą instrukcji START.

Opis implementacji

Kluczowy problem — obsługę zadań współbieżnych — rozwiązano metodą dzielenia czasu procesora na odcinki 40 ms, przyznawane poszczególnym zadaniom przez system operacyjny. Implementowano pięć poziomów priorytetu, każdy dysponujący własną kolejką zadań oczekujących na wykonanie. Przyjęto zasadę, że zadanie z kolejki o niższym priorytecie może otrzymać procesor tylko wtedy, gdy kolejki wyższych priorytetów są puste. Aby uniknąć monopolizowania komputera przez zadania o wysokim priorytecie, przyjęto następujący algorytm postępowania: w chwili zwalniania procesora zadanie nie jest wstawiane na koniec kolejki „swojego" priorytetu, lecz do kolejki priorytetu bezpośrednio niższego niż ten, z którym było uprzednio wykonywane. W ten sposób w kolejnych aktywnych odcinkach czasowych dynamiczny priorytet zadania maleje do poziomu najniższego, po czym cykl powtarza się, począwszy od priorytetu deklarowanego w programie (rys. 2). Każde zawieszone zadanie i uaktywniane przez określone zdarzenie w systemie podejmuje bieg z priorytetem określonym w deklaracji — gwarantuje to szybką reakcję na sygnały ze sterowanego obiektu.

Dostęp zadań współbieżnych do wspólnych zasobów komputera — klawiatury operatora i ekranu — rozwiązano przez wzajemne wykluczanie się dostępu. Rozpoznanie w programie użytkowym instrukcji PRINT albo INPUT łączy się testowaniem znacznika



Rys. 2. Przykładowy podział czasu procesora (zadanie A — priorytet deklarowany 3; zadanie B — priorytet deklarowany 1, priorytet 0 najniższy)

dostępu do konsoli operatorskiej i w przypadku gdy jest ona zajęta, zawiesza zadanie w kolejce typu FIFO. Zawartość tej kolejki jest testowana z chwilą zakończenia realizacji każdej instrukcji PRINT lub INPUT. Jeżeli nie jest ona pusta, to uaktywnione jest kolejne z oczekujących zadań, a znacznik dostępu do konsoli pozostaje w stanie „zajęta".

Aby zidentyfikować zadanie, z którego wyświetlane na ekranie informacje pochodzą, ciąg alfanumeryczny poprzedzony jest zawsze syntaktycznym numerem zadania. Ponadto, w celu uniknięcia blokowania procesora przez oczekiwanie aktywnego zadania na wprowadzenie znaku z klawiatury, system operacyjny wykonuje test gotowości klawiatury przy każdej operacji szeregowania zadań. Pozytywny wynik testu powoduje wczytanie znaku do bufora i w przypadku, gdy jest to CR — wstawienie zadania zawieszonego na instrukcji INPUT do kolejki zadań oczekujących na procesor

Rozkazy komunikacji międzyzadaniowej (SIGNAL, PUT, GET, SEND, RECEIVE) wymagały dynamicznego tworzenia kolejek dostępu na podstawie deklaracji obiektów globalnych. W przypadku instrukcji SEND i RECEIVE, które realizują komunikację i synchronizację między zadaniami metodą tzw. rendez-vous (znana z języka ADA), są to dwie kolejki: zadań oczekujących na odebranie wysłanej przez nie wiadomości oraz zadań oczekujących na odebranie wysłanej wiadowości. Czas oczekiwania może być ograniczany klauzulą TIMEOUT. Drugą opcję powyższych instrukcji (nie definiowaną przez standard), stanowi ELSE. Jej zadaniem jest zapewnienie innej możliwości biegu zadania, gdy „rendez-vous" nie jest natychmiast możliwe. Pozwala to stworzyć m.in. programową konstrukcję zastępującą nieimplementowany blok SELECT:

```
100 DO
105 RECEIVE FROM „ALFA"
    TO X ELSE GOTO 125
    .....
120 EXIT DO
125 RECEIVE FROM „BETA"
    TO Y ELSE GOTO 145
    .....
140 EXIT DO
145 LOOP
```

Implementowany algorytm dopuszcza jednoczesny odczyt tych samych zmiennych przez kilka zadań (instrukcja GET), jednak pojawienie się żądania zapisu (instrukcja PUT) blokuje następne odczyty i zawiesza odpowiednie zadania do czasu zakończenia zapisu.

Zdarzenia, na które oczekują zadania wykonując instrukcję WAIT EVENT, są generowane programowo instrukcją SIGNAL lub przerwaniem zewnętrznym. Powoduje to wstawienie zadania do kolejki oczekiwania na procesor. Efektywna obsługa przerwania zewnętrznego nie jest więc natychmiastowa, lecz przesunięta w czasie, stosownie do priorytetu zadania obsługującego zdarzenie oraz aktualnego obciążenia procesora.

Opisana implementacja IRTB całkowicie spełniła pokładane w niej nadzieje. Narzuty czasowe związane z pracą systemu operacyjnego czasu rzeczywistego wynoszą średnio 5% cza-

su procesora, a stosowanie bardziej rozbudowanych trybów adresacji (m. in. adresacji indeksowej zmiennych wewnętrznych interpretera w bloku operacyjnym każdego zadania) nie spowodowało odczuwalnego spowolnienia interpretacji. Co więcej, badania porównawcze z interpreterm języka BASIC firmy MICROSOFT wykazały zbliżoną szybkość wykonywania programów testowych.

W najbliższym czasie przewidziana jest rozbudowa programu o obsługę sytuacji wyjątkowych (ang. exception). Pozwoli to na zachowanie kontroli nad programem nawet po ich wystąpieniu, w tym również po wystąpieniu błędów spowodowanych współpracą z otoczeniem, a nie wadami programu. Uczyni to z języka INDUSTRIAL BASIC program nadający się do sterowania procesów o szczególnie dużej odpowiedzialności.

WOJCIECH WARSKI
PPZ COMPUTEX, Warszawa

LITERATURA

[1] Iszkowski W., Maniecki M.: Standaryzacja języka BASIC. Informatyka 5—6/1983
[2] IRTB Draft Standard. Raport EWICS TC2 81/3.

Wymiana programów jest dla wielu osób jedynym sposobem zgromadzenia biblioteki. Przekopiowanie programu dla siebie lub w przypadku wymiany z kolegą nie jest niczym zdrożnym. Zresztą, takie wydaje się być też zdanie opinii publicznej na Zachodzie, gdzie problem programowego piractwa jest szeroko dyskutowany. Inaczej jest w przypadku, gdy kopiowanie programów staje się dla kogoś źródłem dochodów. To już jest żerowanie na pracy innych, wyraźnie rozmijające się z zasadami etyki.

Tajemnice programowych piratów – ZX SPECTRUM

Kiedy w 1982 roku zaczęły pojawiać się pierwsze firmowe programy dla ZX SPECTRUM, producenci nie stosowali żadnych zabezpieczeń czy utrudnień w kopiowaniu programów. Programy były zapisywane na taśmie w sposób standardowy — z wykorzystaniem instrukcji SAVE. Aby przekopiować program, wystarczyło odczytać poprzedzający każdy blok nagłówek (17 bajtów zapisanych zaraz za odcinkiem charakterystycznego gwizdu). W nagłówku zawarta jest informacja pozwalająca komputerowi określić, czy następujący po nim blok zawiera program napisany w języku BASIC czy też w kodzie maszynowym, gdzie należy go załadować, czy po załadowaniu program powinien samoczynnie startować itp.

Wydzielenie i odczytanie nagłówka nie jest specjalnie skomplikowane; można się w tym celu posłużyć procedurami zawartymi w pamięci stałej ROM. Procedury takie można odnaleźć posługując się [1]. Jeżeli instrukcją BREAK przerwiemy realizację fragmentu programu napisanego w języku BASIC, to z łatwością można przepisać go ponownie na taśmę magnetofonową (przy zapisie należy pamiętać o podaniu numeru linii, od której rozpoczyna się samoczynna realizacja programu). Pozostałe bloki kopiujemy wykorzystując informację zawartą w nagłówkach (adres początkowy, długość bloku itp.).

Pierwszym utrudnieniem, które zaczęli stosować producenci oprogramowania, było tworzenie „nieprzerywalnych" programów. Program taki można zrealizować wykorzystując zmienną systemową DF-SZ. Określa ona, ile linii u dołu ekranu powinno zostać zarezerwowanych dla wyświetlania komunikatów. Standardowo zmienna ta ma wartość 2 (dwie linie) przechowywaną pod adresem $5C6B_H$ (23659 dziesiętnie). Jeżeli więc wykonana zostanie instrukcja:

POKE 23659,0

to niemożliwe jest wyświetlenie raportu o przerwaniu programu i w rezultacie dalsza praca komputera (trzeba go wyzerować wyłączając zasilanie).

Inny sposób polega na wykorzystaniu 2-bajtowej zmiennej ERR-SP. Jej wartość określa adres początkowy procedury obsługi błędu lub sytuacji specjalnych, takich jak właśnie polecenie BREAK. Jeżeli wartość zmiennej będzie wskazywała na procedurę odmienną od standardowej, to wykonana zostanie właśnie ta procedura (może ona np. niszczyć program). Wartość zmiennej ERR-SP przechowywana jest w dwóch kolejnych lokacjach pamięci o adresach $5C3D_H$ i $5C3E_H$ (dziesiętnie: 23613 i 23614). Żądaną wartość zmiennej ERR-SP można ustawić wykonując dwie instrukcje POKE.

Okazuje się, że programy „nieprzerywalne" można złamać w banalny sposób. Otóż zamiast ładować je instrukcją LOAD, która powoduje natychmiastowy start programu po wczytaniu go przez komputer — należy zastosować instrukcję MERGE. Wczytany w ten sposób program potulnie czeka w pamięci, aż ktoś go wylistuje.

Nieco lepiej zabezpieczone są programy, w których po głównym bloku zawierającym kod maszynowy występują króciutkie bloki nakładek. Wydawałoby się, że dysponując wiedzą o nagłówkach bez problemu ładujmy nakładki. Tymczasem okazuje się, że program odmawia prawidłowej pracy.

Nakładki to zazwyczaj krótkie programy, które wprowadzają pewne modyfikacje kodu, zawartego w głównym bloku. Jednak istota utrudnienia leży w czym innym. ZX SPECTRUM ma specjalne lokacje pamięci (3 bajty), których zawartość jest inkrementowana (zwiększa o jeden) pięćdziesiąt razy na sekundę. Ponieważ jest to częstotliwość ramki obrazu telewizyjnego, zmienną, której wartość przechowywana jest we wspomnianych bajtach nazwano FRAMES (ang. ramki). Zazwyczaj jedna z nakładek ładuje do zmiennej FRAMES określoną wartość, która natychmiast jest inkrementowana. Jeżeli program po załadowaniu samoczynnie startuje, to można oszacować wartość, jaką będzie miała zmienna FRAME w określonym punkcie programu zasadniczego. Tak więc w pewnym momencie realizacji programu (zazwyczaj blisko początku) wartość FRAME porównywana jest ze wzorcem, który może być pobrany z dowolnego miejsca pamięci. Jeżeli nie jest ona prawidłowa, program przechodzi do procedury zerowania systemu. Wystarczy sprawdzić tylko jeden z bajtów FRAME.

Złamanie opisanego zabezpieczenia może polegać na takim przerobieniu zapisanego w języku BASIC programu ładującego, aby wpisywał on również odpowiednią wartość do zmiennej FRAME. Bardziej eleganckim rozwiązaniem jest rozszyfrowanie (z pomocą disasemblera) głównego bloku programu i wyzerowanie (wpisanie rozkazów NOP — kod 00) fragmentu, który wykonuje sprawdzanie wartości FRAMES. Obydwa wymienione sposoby są możliwe dzięki informacjom zawartym w nagłówkach. Dlatego też inwencja producentów oprogramowania poszła w kierunku opracowania metody zapisywania na taśmie bloków, które nie zawierałyby nagłówków.

Zapisywanie bloków informacji nie poprzedzonych nagłówkiem wymaga programowania w kodzie maszynowym, gdyż zastosowane w ZX SPECTRUM instrukcje współpracy z magnetofonem zawsze powodują zapisanie (lub odczyt) nagłówka. Można się przy tym posłużyć procedurami zawartymi w firmowym ROM-ie.

Dla zapisania bloku informacji nie poprzedzonego nagłówkiem, można wykorzystać procedurę rozpoczynającą się od adresu $4C2_H$ (1218 dziesiętnie). Przed jej wywołaniem należy do akumulatora mikroprocesora Z80 wpisać liczbę, w której najbardziej znaczący bit będzie równy 1 (np. FF_H). W rejestrze IX należy umieścić adres początku bloku z ko-

dem maszynowym, który ma być zapisany na taśmie. Natomiast w parze rejestrów DE należy podać liczbę bajtów w bloku.

Przed blokiem informacji przez ok. 2 sekundy zapisywany jest ciąg impulsów o długości 2168 okresów zegara procesora (T). Pozwala on na odpowiednie ustawienie układów automatyki w magnetofonie, a przy odczytywaniu informuje układy wejściowe mikrokomputera, że za moment rozpocznie się informacja cyfrowa. Na końcu ciągu zapisywane są dwa impulsy synchronizujące: przez 667 okresów T wyjście „mikrofon" jest wyłączone, a następnie przez 735 okresów T jest ono włączone. Jako pierwszy bajt informacji zapisywany jest znacznik określający, czy dany blok jest nagłówkiem (wartość 00) czy blokiem danych (wartość FF_H). Na końcu bloku zapisywany jest bajt kontrolny. Powstaje on przez wykonywanie rozkazu XOR kolejnych bajtów danych z bajtem kontrolnym. Jako wartość początkową bajtu kontrolnego przyjmuje się 00 lub FF_H — odpowiednio do wartości znacznika typu bloku.

Bloki bez nagłówka muszą być wczytywane przez procedurę napisaną w języku maszynowym. Można w tym celu wykorzystać procedurę umieszczoną w ROM-ie od adresu 556_H (1366 dziesiętnie). Przed jej wywołaniem należy do akumulatora wpisać znacznik typu FF_H (nagłówek miałby wartość 00). W rejestrze IX należy podać adres, od którego dane mają być ładowane do pamięci. W parze rejestrów DE podaje się liczbę bajtów w bloku, który będzie ładowany. Ważne jest też, aby znacznik przeniesienia (ang. CARRY FLAG) był w stanie logicznym 1. Dowolny program, który ma być ładowany do komputera przez „normalnego" użytkownika (a więc instrukcją LOAD) musi zawierać blok zapisany w języku BASIC, zawierający nietrudną do zlokalizowania procedurę w kodzie maszynowym lub przynajmniej jeden blok z programem w języku maszynowym poprzedzony nagłówkiem. Pozostaje więc skorzystać z disasemblera, aby dowiedzieć się, jakie wartości ładowane są do poszczególnych rejestrów mikroprocesora przed wywołaniem procedury wczytującej blok bez nagłówka.

Aby zabezpieczyć się przed „niepowołanym" dostępem do wspomnianego bloku z kodem maszynowym, wymyślono kolejną sztuczkę: samostartujący blok napisany w języku maszynowym. Jest to możliwe przez odpowiednie wykorzystanie zmiennych systemowych komputera. Zapisywany blok kodu powinien zawierać oprócz programu również przestrzeń zmiennych systemowych oraz przestrzeń, w której przechowywany jest kod pośredni programów w języku BASIC. Ponadto należy również zapisać obszar stosu, który zajmuje miejsce poniżej adresu pamiętanego w zmiennej RAMTOP. Tak zapisany kod spowoduje wznowienie realizacji programu od miejsca, w którym została ona przerwana wykonaniem instrukcji SAVE. Przykładowo — zapis samoczynnie startującego kodu maszynowego można wykonać w następujący sposób:

```
10 SAVE „NAME" CODE 23552,4000
20 RAND USR 27000
```

Adres 23552 ($5C00_H$) określa początek obszaru zmiennych systemowych. Załadowany w ten sposób kod rozpocznie pracę od realizacji instrukcji RAND USR.

Okazuje się, że nawet tak wymyślne zabezpieczenie nie stanowi przeszkody dla piratów. Jeżeli program nie jest zbyt długi, należy odpowiednio zredukować wartość zmiennej RAMTOP — tak, aby można było go załadować powyżej wskazywanej przez nią granicy. Uniemożliwia to samoczynny start i pozwala na wykonanie dowolnych czynności (z ponownym zapisaniem na taśmę włącznie). Jeśli jednak program jest na tyle obszerny, że zmiennej RAMTOP nie można nadać tak niewielkiej wartości, to sposobem na przerwanie realizacji programu jest spowodowanie generacji komunikatu o błędzie zaraz po załadowaniu programu. Powoduje to przejście od interpretera języka BASIC, co wystarcza, by móc zająć się wpisanym programem. Dla generacji błędu przy odczycie wystarczy podać zbyt dużą długość bloku. Generacja błędu nie może tu zostać zablokowana w omawiany wcześniej sposób (poprzez zmienne DF-SZ lub ERR-SP), gdyż zapis poprzez instrukcję SAVE generuje komunikat w dolnej części ekranu.

Pozostaje jednak jeszcze przypadek, gdy blok kodu wypełnia cały obszar pamięci RAM od adresu 4000_H do $FFFF_H$. Otóż powstały już programy ładujące, umieszczane... w obszarze zmiennych systemowych, które ładują nawet bloki bez nagłówka w cały obszar pamięci, ale z wyłączeniem miejsca, które same zajmują.

Na zakończenie dodajmy, że programy można też kopiować z magnetofonu na magnetofon, choć oczywiście nie dostarcza to takiej satysfakcji jak łamanie zabezpieczeń. Metoda ta wymaga jednak sprzętu wysokiej klasy o szerokim paśmie przenoszenia (przez obcięcie wyższych częstotliwości „prostokątne" impulsy cyfrowe ulegają zaokrągleniu). Ważna jest też w tym przypadku precyzyjnie dobrana szybkość przesuwu taśmy. Z marnym pasmem przenoszenia krajowych magnetofonów można sobie poradzić włączając między magnetofony układ formowania impulsów.

AJP

LITERATURA

[1] Logan I., O'Hara F.: The Complete SPECTRUM ROM disasembly. Melbourne House Publishers Ltd., 1983.

Czytelnikom, którzy od tego numeru rozpoczęli stały kontakt z mikroKLANEM winni jesteśmy kilka wyjaśnień na temat poniższego cyklu. W Akademii mikroKLANU nie uczymy programowania od podstaw. Zainteresowanych wprowadzeniem w programowanie odsyłamy do instrukcji obsługi posiadanego komputera lub do biblioteki, gdzie z pewnością znajdą kilka książek o programowaniu w języku BASIC. Wielbicieli kursów w odcinkach odsyłamy do „Horyzontów Techniki", które podjęły próbę uporządkowania prowadzonego tam kursu.

W Akademii mikroKLANU staramy się przekazać Czytelnikom umiejętność eleganckiego programowania, czyli takiego, które pozwoli nam samym (i innym!) wielokrotnie wykorzystać pracę włożoną w napisanie programu.

Akademia mikroKLANU (4)

Kursor

Podział programu na moduły bardzo ułatwia opracowanie całości, a w przypadku długich programów jest właściwie jedynym sposobem przetestowania. Krótki „programik" łatwo objąć całościowo — łatwiej więc wprowadzać poprawki i usuwać błędy.

Język BASIC nie bardzo nadaje się do strukturalizacji, Zobaczmy jednak, co można zrobić. Pierwszym krokiem było już umieszczanie komentarzy w innych liniach niż instrukcje. Ten sam sposób możemy zastosować do podziału programu na moduły: niech wejście i wyjście mają numery linii rzędu dziesiątek, procedura główna rzędu setek, a procedury poboczne niech zaczynają się od numerów będących wielokrotnościami tysięcy. Tak naprawdę ułatwienie pisania programu poprzez jego strukturalizację zauważymy jednak, gdy jakaś część programu będzie się powtarzała (np. obliczanie wartości skomplikowanego wyrażenia lub gdy stosujemy podobne procedury wejścia-wyjścia w kilku miejscach w programie). Wtedy nasze gotowe, przetestowane moduły, możemy bardzo łatwo umieścić w programie, a nawet wykorzystać w innych programach.

Nasz szkolny program IV jest taką właśnie gotową procedurą, służącą do przesuwania kursora po całym polu ekranu dla wskazania pewnego położenia. Procedura ta została napisana w języku BASIC dla komputera ZX SPECTRUM. Ewentualne zmiany procedury w celu przy-

stosowania jej do innych komputerów są bardzo proste i ograniczają się do zmiany wielkości obrazu (zmienne L oraz H oznaczające szerokość i wysokość pola) oraz do zastąpienia instrukcji INKEY$ odpowiednią instrukcją odczytującą wciśnięty klawisz (np. w komputerach COMMODORE taką instrukcją jest GET, przy czym należy użyć zmiennej łańcuchowej, tzn. GET B$ i porównywać wczytany znak z opisami umieszczonymi na klawiszach). Należy też oczywiście zmienić początek pamięci obrazowej, czyli zmienną TP.

A oto procedura:

```
LIST

1000 REM PROCEDURA SUWANIA KURSOREM (JASNY KWADRAT)
1010 LET TP=22528
1011 REM TP JEST POCZATKIEM PAMIECI VIDEO (LEWY GORNY ROG)
1020 LET L=32
1030 LET H=22
1031 REM L ORAZ H OZNACZAJA ILOSC KOLUMN I WIERSZY OBRAZU VIDEO
1040 LET XG=0
1050 LET YG=0
1051 REM XG ORAZ YG TO AKTUALNE POLOZENIE KURSORA
1060 LET A=PEEK TP+XG+YG*L
1061 REM A ZAPAMIETUJE ELEMENT,GDZIE POJAWIA SIE KURSOR
1070 POKE TP+XG+YG*L,134
1071 REM KOD 134 OZNACZA MIGAJACY KWADRAT
1080 LET XX=XG
1090 LET YY=YG
1091 REM ZAPAMIETANIE OSTATNIEGO POLOZENIA KURSORA
1100 IF INKEY$="" THEN GOTO 1100
1101 REM KOMPUTER CZEKA NA WCISNIECIE KLAWISZA
1110 IF INKEY$="8" THEN LET XG=XG+1-INT((XG+1)/L)*L
1120 IF INKEY$="5" THEN LET XG=XG-1+(1-SGN(XG))*L
1130 IF INKEY$="6" THEN LET YG=YG+1-INT((YG+1)/H)*H
1140 IF INKEY$="7" THEN LET YG=YG-1+(1-SGN(YG))*H
1141 REM PRZESUNIECIA KURSORA ODPOWIADAJA NATURALNYM KLAWISZOM SPECTRUM
1150 IF INKEY$="Y" THEN RETURN
1151 REM WCISNIECIE RETURN (LUB Y) POWODUJE POWROT DO GLOWNEGO PROGRAMU
1160 POKE TP+XX+YY*L,A
1161 REM WYPELNIENIE POPRZEDNIEGO POLOZENIA KURSORA STARYM ELEMENTEM
1170 GOTO 1060
READY.
```

Jeszcze pewna uwaga praktyczna, ułatwiająca eleganckie pisanie programów. Kiedy ostatni raz Czytelnicy użyli funkcji SGN? Jest to standardowa funkcja w języku BASIC i jej wykonanie jest bardzo szybkie. W naszym szkolnym programie (linie 1120 oraz 1140) dzięki SGN udało się uniknąć skoków warunkowych — można powiedzieć, że funkcja SGN działa tu jako element decyzyjny. Takie sztuczki, choć nieco zaciemniają algorytm programu, ale za to r z e c z y w i ś c i e go skracają. Trzeba tylko je dobrze udokumentować!

Zachęcamy Czytelników do poeksperymentowania — komputera n i e m o ż n a zepsuć z klawiatury... Warto sprawdzić, co się stanie, gdy opuścimy linijkę 1160. A co będzie, jeśli opuścimy nasz trick w linii 1120?

JAKUB TATARKIEWICZ

- „Prace" nad powołaniem osobnego pisma „mikroKLAN" w toku, chociaż:
  - — brakuje papieru
  - — drukarnie robią bokami i bez mikroKLANU
  - — decydenci nie chcą słyszeć o wydawaniu zezwoleń na nowe czasopismo bo... obowiązuje moda na książki
  - — wydawnictwo nie może dać nowych etatów ze względu na złowieszczy FAZ. Są jednak ciągle tacy co wierzą, że pismo powstanie.
- Zniesienie bariery celnej na mikrokomputery przyniosło spodziewany raptowny spadek cen wolnorynkowych! ZX SPECTRUM 48K można było dostać za 100 ... 120 tys. zł., a COMMODORE C64 za ok. 160 tys. zł. Niektórzy pozbywali się komputerów za „zielone", w kwotach zbliżonych do obowiązujących na Zachodzie. Sytuacja zapewne nieco zmieniła się od 1 kwietnia, tzn. po zablokowaniu nieudokumentowanych wpłat na konta dewizowe typu A.

µK

prowadzi:
Andrzej J. Piotrowski
tel. dom.: 48-22-85

Nowej wersji systemu CP/M nie wróży się na Zachodzie długiego życia, podobnie jak mikrokomputerom wykorzystującym mikroprocesory 8-bitowe. Z zachodniej perspektywy jest to jakby proteza — pozwalająca sprostać rosnącym wymaganiom użytkowników i przetrwać do momentu, gdy „szesnastki" staniejąa do kilku dolarów.

My mamy jednak inną perspektywę i — poza nielicznymi próbami składania IBM PC przez firmy polonijne — nic nie wróży rychłego upowszechnienia się w kraju mikrokomputerów 16-bitowych. CP/M 3+ jest więc może dla nas deską ratunku. Przekonstruowanie oferowanych w kraju mikrokomputerów pod kątem nowego systemu nie jest takie trudne. Jeśli mikrokomputery z serii MSX staniejąa na tyle, że odniosą wśród osób prywatnych podobny sukces jak ZX SPECTRUM, CP/M 3+ miałby na naszym rynku sporą szansę.

CP/M 3+

Najpopularniejszym systemem operacyjnym dla mikrokomputerów 8-bitowych (wykorzystujących Z80 lub 8080) jest system CP/M — wersja 2.2. Dysponuje on niedużą pamięcią operacyjną (maks. 64 KB), co w poważnym stopniu ogranicza jego możliwości. Obecnie firma DIGITAL RESEARCH opracowała nową wersję systemu, oznaczoną **CP/M — Plus wersja 3.0**, która może zarządzać znacznie większym obszarem pamięci operacyjnej. Osiągnięto to nie tylko na drodze rozwiązań programowych, lecz przede wszystkim — przez zastosowanie nowego sposobu połączenia bloków pamięci operacyjnej.

Możliwa jest dzięki temu całkowita zmiana koncepcji pracy systemu, co czterokrotnie zwiększa jego wydajność. System jest w stanie obsługiwać 16 jednostek pamięci na dyskach elastycznych. Ponadto wprowadzono wiele rozwiązań ułatwiających eksploatację systemu.

Dodatkową zaletą systemu CP/M-Plus jest możliwość wykonywania pod jego nadzorem prawie wszystkich programów działających pod systemem CP/M-2.2. Wyjątek stanowią programy odwołujące się bezpośrednio do procedur w module BIOS (Basic Input Output System) wersji 2.2.

Organizacja pamięci operacyjnej

Możliwość bezpośredniego adresowania pamięci przez procesor 8-bitowy jest ograniczona do 64 KB. Limituje to maksymalną długość segmentów programów użytkowych, liczbę obsługiwanych urządzeń we-wy, szybkość działania, itp.

Projektanci systemu CP/M-Plus 3.0 znaleźli takie rozwiązanie problemu, które po stronie sprzętu (ang. hardware) nie wymaga skomplikowanych konstrukcji. Polega ono na zastosowaniu specjalnego układu, pozwalającego na równoległe [1]) połączenie ze sobą wielu bloków pamięci operacyjnej. W ten sposób można uzyskać pamięć operacyjną złożoną z 16 bloków po 64 KB każdy, co daje łączny obszar o wielkości 1 MB.

Minimalny obszar pamięci operacyjnej, umożliwiający korzystanie ze wszystkich funkcji systemu operacyjnego, wynosi 128 KB (dwa bloki). Możliwa jest także instalacja systemu w pamięci 64 KB, wiąże się to jednak z rezygnacją z wielu funkcji oraz znacznym obniżeniem wydajności.

Z każdego z połączonych bloków pamięci wydzielony zostaje obszar o wielkości 4 do 16 KB, przez co uzyskuje się jeden obszar, traktowany przez system jako spójny. Obszar ten jest wykorzystywany w procesie zarządzania pamięcią operacyjną oraz pośredniczy przy realizacji odwołań programów do systemu operacyjnego.

Kompletny system zajmuje 21 KB w pierwszym bloku pamięci. Obszar przeznaczony na wczytywanie programów jest wydzielony z 1 i 2 bloku i wynosi 60 KB. Istnienie pozostałych (do 14) bloków pamięci operacyjnej uzależnione jest od liczby podłączonych jednostek dyskowych oraz od planowanej wydajności systemu. Bloki te są wykorzystywane wyłącznie przy obsłudze jednostek dyskowych.

Obsługa pamięci dyskowych

Głównym czynnikiem obniżającym szybkość działania systemów mikrokomputerowych jest współpraca z jednostkami sterującymi dysków elastycznych. Straty czasowe rosną wraz z liczbą dostępów do tych pamięci. W systemie CP/M-Plus problem ten rozwiązywany jest różnymi metodami, sprowadzającymi się jednak do utrzymania liczby dostępów na jak najniższym poziomie. Odbywa się to kosztem poświęcenia przeważającej części pamięci operacyjnej na obsługę pamięci dyskowych.

Jedna z zastosowanych metod polega na wczytywaniu do pamięci operacyjnej całej tablicy adresowej dyskietki (jej długość wynosi często wiele KB). W ten sposób na przeszukanie całej tablicy wymagany jest tylko jeden dostęp do dyskietki. Metoda ta jest określana w języku angielskim jako Directory Caching.

Druga metoda polega na przypisaniu dużych obszarów pamięci operacyjnej dla poszczególnych jednostek dyskowych na przetwarzanie danych pobieranych z tych jednostek (ang. cache memory). Umożliwia to operowanie na dużych częściach zbiorów zamiast na pojedynczych rekordach, co także obniża liczbę dostępów do dyskietek. W momencie gdy program żąda dostępu do określonego rekordu, system operacyjny sprawdza w pierwszej kolejności, czy rekord ten znajduje się już w pamięci operacyjnej. Dopiero gdy go tam nie znajdzie, następuje pobranie z dyskietki odpowiedniej części zbioru.

Jak widać, system CP/M-Plus kieruje się zasadą, że w trakcie jednego dostępu należy czytać i zapisywać w pamięci dyskowej jak najwięcej danych. Zasada ta jest stosowana także w przypadku tworzenia nowego zbioru. Rekordy tworzone w pamięci operacyjnej przepisywane są na dyskietki dopiero po zapełnieniu się obszaru przeznaczonego na ten cel. Maksymalna liczba danych przepisywanych w ten sposób wynosi 16 KB. Podobna transmisja wymagałaby w systemie CP/M-2.2 128 dostępów do pamięci dyskowej.

Dodatkowe usprawnienia systemu

W porównaniu do wersji 2.2, w systemie CP/M-Plus 3.0 wprowadzono ok. 130 nowych funkcji — ułatwiających jego użytkowanie.

Między innymi zautomatyzowano procedurę zmiany dyskietek. System sam rozpoznaje pojawienie się nowej dyskietki i odnotowuje ten fakt w odpowiedniej tablicy, nie czekając na dyrektywę operatorską.

Innowacją jest także wprowadzenie hasłowej ochrony całych dyskietek lub poszczególnych zbiorów. Ochronie może podlegać każda z podstawowych operacji przeprowadzanych na zbiorach, tzn.: zapis, odczyt i zamazywanie informacji. Wprowadzono także wiele dodatkowych funkcji ułatwiających obsługę zbiorów dyskowych.

Rozbudowano również obsługę błędów przez wprowadzenie wyczerpujących komunikatów o błędach. Umożliwiono korygowanie niektórych błędów w drodze dialogu z systemem operacyjnym (pytanie systemu, odpowiedź użytkownika). Dopiero przy braku możliwości bieżącej korekcji błędu następuje całkowite przerwanie wykonywania programu.

Na uwagę zasługuje także fakt zwiększenia liczby programów usługowych (ang. utilities) działających w systemie. Aby uprościć obsługę tych programów, wprowadzono funkcję HELP, która umożliwia korzystanie z nich bez potrzeby używania podręcznika.

WITOLD WOŹNIAK
Warszawa

[1]) Procesor zawsze „widzi" przestrzeń 64KB. Rozwiązanie polega na przyłączaniu do procesora (pod kontrolą systemu) żądanego w danym momencie bloku — przyp. AJP

Wśród opisywanych w mikroKLANIE bloków funkcjonalnych wchodzących w skład mikrokomputera brakowało do tej pory rozwiązań umożliwiających wyświetlanie informacji na ekranie telewizyjnym. Wypełniając tę lukę, rozpoczynamy cykl publikacji o sterownikach CRT. Zaczynamy od rozwiązania uproszczonego, zrealizowanego na łatwo dostępnych elementach — układach TTL.

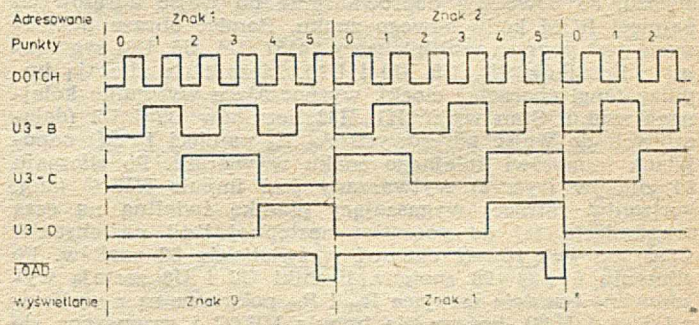
Prezentowane rozwiązanie umożliwia zapisywanie do 64 znaków w wierszu (więcej niż wiele popularnych komputerów domowych). Chcąc zapisać większą liczbę — trzeba by wykorzystać profesjonalny monitor ekranowy — natomiast opisaną konstrukcję można podłączyć do nieznacznie tylko zmodyfikowanego telewizora (dobudowa wejścia video). Warto jeszcze dodać, że opisane rozwiązanie można w przyszłości zmodyfikować (rozbudowując generator znaków o pamięć RAM) — tak, aby możliwe było wykorzystywanie pseudografiki.

Alfanumeryczny sterownik monitora telewizyjnego (1)

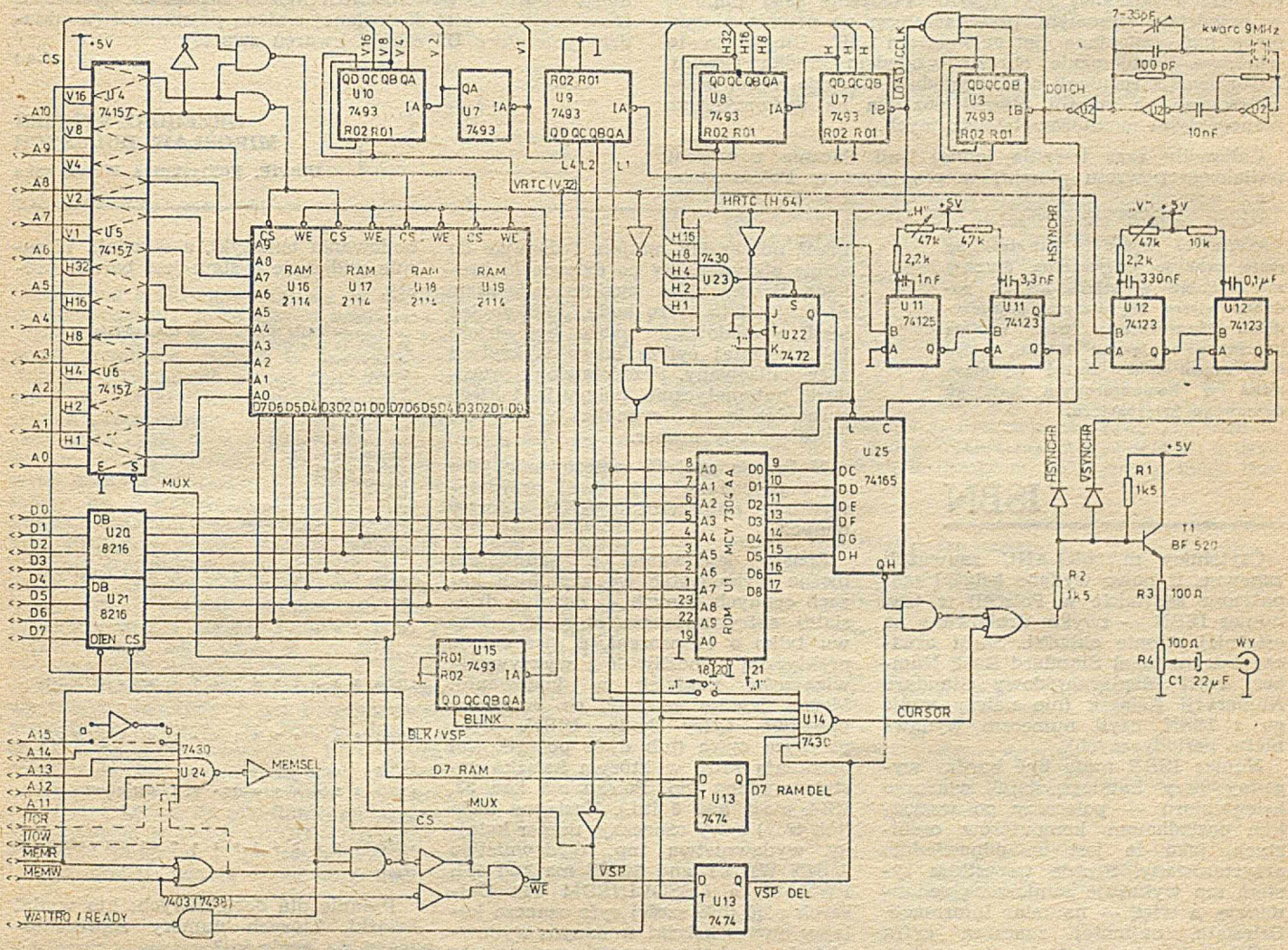
Opisany w tekście sterownik (rys. 1) umożliwia wyświetlanie na ekranie monitora telewizyjnego 2048 znaków alfanumerycznych, rozmieszczonych w 32 wierszach o 64 znakach. Wyświetlane na ekranie znaki wpisywane są przez mikroprocesor do wydzielonego zespołu 2 KB pamięci RAM, stanowiącego pamięć obrazu. Adresy poszczególnych bajtów pamięci obrazu odpowiadają określonym miejscom na ekranie — tzn. numerom kolumn i wierszy. Znaki zapisywane są w 7-bitowym kodzie ASCII. Ósmy bit (nie wykorzystywany w tym kodzie) stosowany jest do włączania kursora.

Znaki wyświetlane są na ekranie monitora przez rozjaśnienie odpowiednich punktów w polu obejmującym osiem kolejnych linii po sześć punktów każdej.

Zasadniczą funkcję przetwarzania zawartości pamięci obrazu na sygnały rozświetlające punkty ekranu realizuje specjalny układ pamięci stałej ROM — tzw. generator znaków (U1). Na wejścia adresowe A_3...A_9 tego układu podawany jest z pamięci obrazu kod wyświetlanego znaku. Na



Rys. 2. Przebiegi czasowe licznika punktów U3

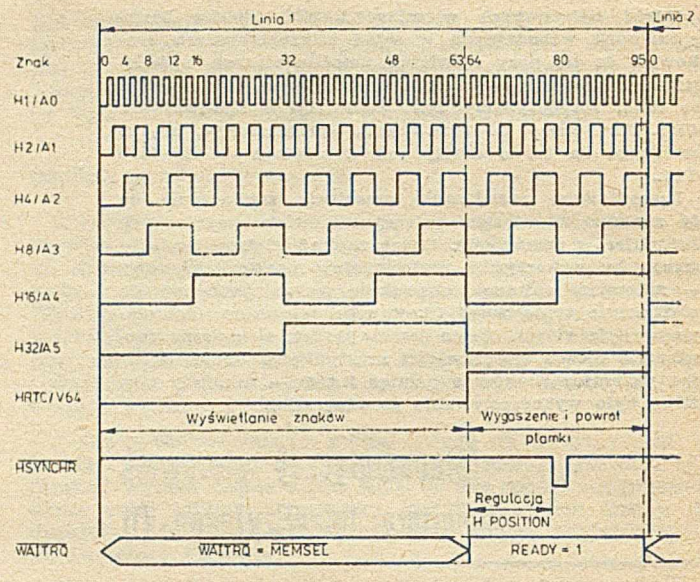


Rys. 1. Schemat ideowy sterownika monitora telewizyjnego

wejścia adresowe $A_0...A_2$ podawany jest numer wyświetlanej linii danego znaku. Sygnały na wyjściach $D_0...D_7$ odpowiadają rozkładowi punktów świetlnych w aktualnie wyprowadzanej, linii znaku. Sygnały te są podawane kolejno (szeregowo) do wyjścia wizyjnego przez rejestr przesuwny 74165 (U25). Wyprowadzanie sygnałów jest taktowane przez tzw. generator punktów (jego częstotliwość odpowiada czasowi wyświetlania punktu na ekranie). Dane z pamięci ROM wpisywane są do rejestru przesuwnego w trakcie trwania impulsu LOAD/-CCLK z licznika U3, odliczającego sześć punktów (jedna linia znaku). Impuls ten formowany jest przez współpracującą z licznikiem U3 bramkę (rys. 2).

Przejście do wyświetlania kolejnego znaku (ta sama linia) wymaga powiększenia o 1 adresu podawanego do pamięci obrazu, co powoduje doprowadzenie do wejść adresowych pamięci ROM kodu nowego znaku. Pamięć obrazu adresowana jest poprzez multipleksery 74157 (U4, U5, U6) sygnałami z liczników U7, U8, U9 i U10. Liczniki te zliczają impulsy (ang. character clock), wyznaczające wysyłanie kolejnego znaku. Stan wyjść H1...H32 liczników U7 i U8 (dołączonych do wejść adresowych $A_0...A_5$ pamięci RAM) odpowiada numerowi kolejnego znaku w wierszu. Po zliczeniu 64 znaków (rys. 3) wytwarzany jest impuls HRTC (ang. horizontal retrace) wygaszający plamkę świetlną na czas przesunięcia jej do początku następnej linii na ekranie. Czas ten jest równy czasowi wyświetlania 32 znaków. Po zliczeniu 64+32=96 znaków, liczniki U7 i U8 zostają wyzerowane (wejścia zerujące R_{01} i R_{02} połączone są z wyjściami H64 i H32), kończy się impuls HRTC i rozpoczyna się liczenie znaków następnej linii obrazu. Impuls HRTC wyzwala uniwibratory U11 (74123), wytwarzające impuls synchronizacji poziomej HSYNCHR/. Położenie tego impulsu w przedziale czasowym, przeznaczonym na powrót plamki może być zmieniane potencjometrem „H". Umożliwia to właściwe usytuowanie obrazu względem środka ekranu. Zastosowanie regulacji oraz względnie długiego czasu, przeznaczonego na powrót plamki pozwala na wykorzystanie różnego typu monitorów telewizyjnych.

Całkowity czas trwania jednej linii (łącznie z czasem poziomego powrotu plamki) w przyjętym w Polsce standardzie telewizyjnym jest równy 64 µs. Częstotliwość generatora punktów U2 powinna wynosić zatem 6×(64+32):64 µs=9 MHz. Częstotliwość ta może różnić się od wymaganej o ok. ±10%. W przypadku trudności z uzyskaniem odpowiedniego kwarcu z powodzeniem można zastosować generator RC o częstotliwości dostrajanej zmianą pojemności (w układzie generatora U2 zaciski kwarcu zwarte).



Rys. 3. Przebiegi czasowe liczników znaków

(c.d.n.)

GRZEGORZ STĘPIEŃ
MIROSŁAW DOLEŻYCH
IBSPiE, Politechnika Warszawska

Króciutkie procedury (niestety na razie brakuje miejsca na dłuższe) zamieszczane w mikroKLANIE podobno cieszą się dużym powodzeniem. Prezentujemy więc kolejną procedurę w języku BASIC, która tym razem może zostać wykorzystana w komputerowym katalogu domowego księgozbioru.

ISBN

Czytelnicy mikroKLANU zauważyli zapewne, iż nowo wydane książki zaopatrzone są (także w Polsce!) w tzw. numer ISBN — zwykle drukowany na ostatniej stronie okładki. Skrót oznacza International Standard Book Number, czyli Międzynarodowy Standard Numeracji Książek (nie należy mylić go z ISSN, czyli numerem wydawnictw periodycznych).

Numer ISBN może być bardzo wygodny przy katalogowaniu własnego księgozbioru — polecamy go wszystkim posiadaczom komputerów osobistych, jako że jest on odpowiednio przystosowany. Przede wszystkim system ten wykazuje zamierzoną redundancję, a więc — nadmiar informacji. Dziesiąta, najmniej znacząca cyfra ISBN jest cyfrą kontrolną (sprawdzającą): mnożąc kolejne cyfry ISBN przez numer ich miejsca i dodając do siebie otrzymujemy po dziewięciu krokach liczbę, która modulo jedenaście daje dziesiątą cyfrę kodu. Jakikolwiek błąd w kodowaniu może być dzięki tej dziesiątej cyfrze łatwo wykryty i zasygnalizowany. Przedstawiona procedura automatycznie dokonuje takiego sprawdzenia i sygnalizuje błędnie wprowadzony numer.

Może ona znaleźć zastosowanie jako część procedury wejściowej przygotowywania najprzeróżniejszych zbiorów danych.

Sposób kodowania z nadmiarem można wykorzystać we wszelkich bazach cennych danych — dodanie dziesiątej cyfry zwiększa objętość zestawu tylko o 11 procent, a gwarantuje praktyczną możliwość wykrywania większości pomyłek w kodowaniu. Można jeszcze zapytać, co oznaczają pozostałe cyfry kodu ISBN. Otóż pierwsza cyfra (lub dwie początkowe) oznaczają kraj, w którym książka została wydana (np. Polska ma kod 83, USA mają kod 0 itd.). Następne dwie (lub trzy) cyfry oznaczają numer kolejny wydawnictwa (np. wydawnictwo JOHN WILEY and SONS ma kod 471, PWN 01, a OSSOLINEUM 04). Dalszych pięć (lub sześć) cyfr oznacza kolejny numer książki w danym wydawnictwie i niczym specjalnym się nie wyróżnia (nie ma tu możliwości zakodowania tematyki, a szkoda!). W sumie widać, że system jest bardzo prosty.

```
10 DIM A(10)
20 PRINT "WPROWADZ NUMER ISBN (OD LEWEJ DO PRAWEJ)"
30 FOR I=10 TO 1 STEP -1
40 INPUT A(I)
50 NEXT I
60 PRINT "LICZBA ";
70 FOR I=10 TO 1 STEP -1
80 PRINT A(I);
90 NEXT I
100 S=0
110 FOR I=1 TO 10
120 S=S+A(I)*I
130 NEXT I
140 S=S-INT(S/11)*11
150 IF S<>0 THEN PRINT " NIE";
160 PRINT " JEST W KODZIE ISBN"
170 END
READY.

LICZBA 8 3 7 1 5 8 2 6 1 5  JEST W KODZIE ISBN

READY.

LICZBA 2 0 8 6 4 3 3 0 1 0  JEST W KODZIE ISBN

READY.

LICZBA 8 3 7 1 5 8 2 6 5 6  JEST W KODZIE ISBN

READY.

LICZBA 9 6 3 0 5 3 1 9 4 1  JEST W KODZIE ISBN

READY.

LICZBA 3 5 4 0 1 0 7 4 9 5  JEST W KODZIE ISBN

READY.
 5
LICZBA 5 0 0 2 0 0 0 1 9 5  NIE JEST W KODZIE ISBN

READY.
```

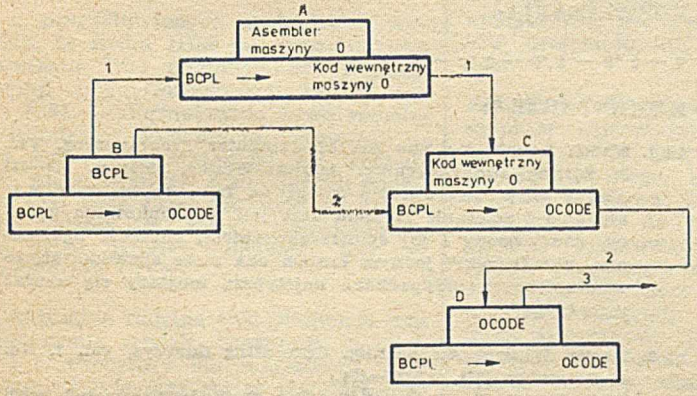
Pytanie dla dociekliwych: kto wydał książki, których numery zaprezentowano na wydruku?

JAKUB TATARKIEWICZ

dokończenie ze str. 12

na drugiej (gdzie znajduje się translator). Mając działający translator TRAN1 napisany w języku asemblera, można napisać w języku BCPL drugi translator o tym samym przeznaczeniu, a po uzyskaniu jego kodu OCODE — przy użyciu TRAN1 wygenerować TRAN2 (na ogół bardziej optymalny) i dalsze translacje programów użytkowych prowadzić tylko za jego pomocą.

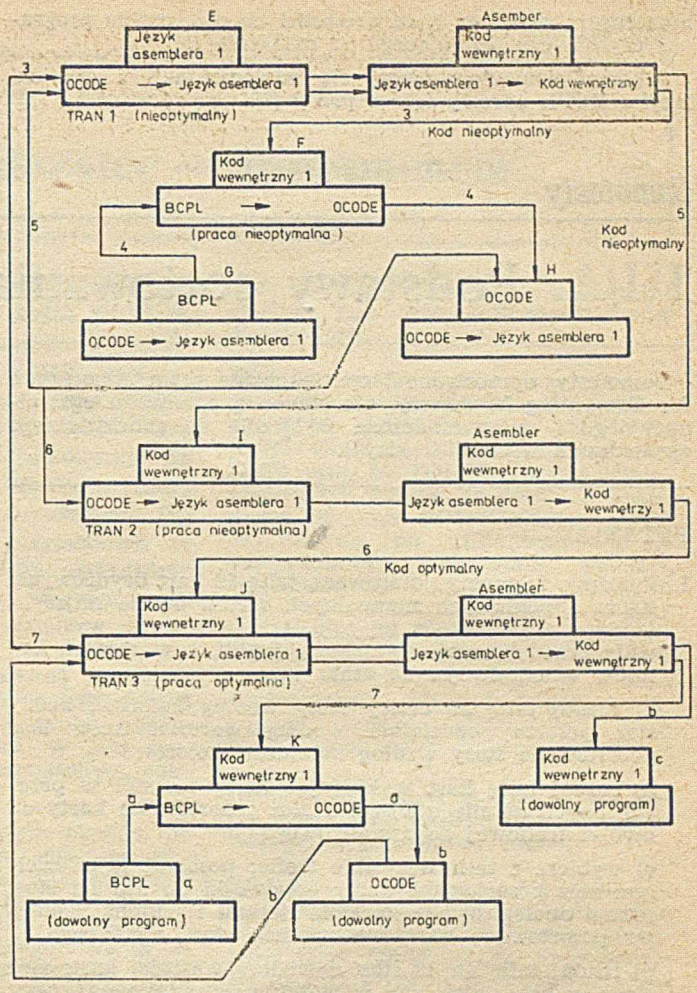
Gdy nie istnieje praktycznie dostępny kompilator języka BCPL, to pierwszym krokiem jest napisanie i uruchomienie najprostszego programu kompilującego na maszynie 0 w jej języku asemblera [1]. Program, który nazwiemy A tłumaczy tekst w języku BCPL na kod wewnętrzny maszyny 0. Następnym krokiem jest napisanie w języku BCPL programu kompilatora tego języka na kod OCODE. Nazwiemy ten program literą B [1]. Tekst programu B, traktowany jako tekst wejściowy, może być przetłumaczony (skompilowany) działającym już programem A (przebieg 1 na rysunku 1). Uzyska się wtedy — także działający na maszynie 0 — program kompilatora z języka BCPL na kod OCODE. Nazwiemy go programem C. Program ten może przekompilować (przebieg 2) na maszynie 0 tekst kompilatora BCPL-OCODE napisany w języku BCPL (B). W ten sposób uzyskuje się także kompilator, ale już opisany w kodzie OCODE. Nazwiemy go D. Na tym kończy się wykorzystanie maszyny 0. Schemat generacji kodu OCODE kompilatora przedstawiono na rysunku 1.



Rys. 1. Generacja kodu OCODE kompilatora BCPL

Następne kroki wykonuje się na maszynie docelowej, nazwanej maszyną 1. W języku asemblera maszyny 1 należy napisać program E, tłumaczący tekst z kodu OCODE na jej język asemblera. Będzie to tzw. translator nieoptymalny. Jego nieoptymalność polega zarówno na sposobie działania, jak i na uzyskiwanym produkcie wyjściowym. Pozwala on jednak przetłumaczyć (przebieg 3 z rysunku 2) tekst zapisany w OCODE, będący programem kompilatora zapisanego w tym kodzie. Niezbędne jest oczywiście użycie standardowego asemblera maszyny 1. Uzyskuje się program F działający na maszynie 1, którego praca jest nieoptymalna wskutek użycia translatora E generującego nieoptymalny kod. Następnie należy napisać ponownie w języku BCPL program translatora kodu OCODE na język asemblera maszyny 1 (program G), który po skompilowaniu go przez kompilator F pozwala uzyskać ten sam program, ale napisany w kodzie OCODE i nazwany H. Ten nowy program potraktowany jako tekst w OCODE może zostać przetłumaczony przy użyciu programu E (przebieg 5), który wytwarza nieefektywnie pracujący program w kodzie wewnętrznym maszyny 1 (po przetłumaczeniu standardowym asemblerem maszyny 1). Jest to druga wersja translatora OCODE na język asemblera, który wprawdzie działa nieefektywnie, ale pozwala uzyskać kod wewnętrzny dla maszyny 1 działający efektywnie. Nazwiemy go I. Program ten służyć będzie do przetłumaczenia programu H (przebieg 6) zapisanego w kodzie OCODE. Uzyskany wskutek tego tłumaczenia program J (po zastosowaniu asemblera maszyny 1) jest trzecią wersją translatora kodu OCODE na język

[1]) Maszynę 0 wybiera się w ten sposób, aby jej istniejące oprogramowanie maksymalnie ułatwiało fazę wstępną



Rys. 2. Optymalizacja kompilatora i translatora na maszynie docelowej

asemblera; generuje zarówno optymalnie działający kod, jak i sam działa optymalnie. Jednym z pierwszych programów tłumaczonych przy użyciu translatora J jest tekst opisujący w języku OCODE kompilator BCPL-OCODE (program D) otrzymany jeszcze na maszynie 0 (przebieg 7). Uzyskany w ten sposób i działający optymalnie na maszynie 1 kompilator BCPL-OCODE nazwiemy K. Na tym kończy się implementacja języka BCPL na maszynie 1. Na rysunku 2 przedstawiono pełen schemat drugiego etapu generacji kompilatora i translatora. Przedstawiono na nim także przebieg typowej kompilacji i translacji programów użytkowych na maszynie 1. Program użytkowy „a" napisany w języku BCPL jest kompilowany programem K, a uzyskana postać w OCODE (program „b") jest z kolei tłu-

```
GLOBAL      $( OUTPUT:4 ; PACKSTRING:20 $)

MANIFEST    $( FILE=0 ; BUFFER=1
               POINTER=2 ; TRANSFER=3
               BUFFERSIZE=120   $)

STATIC      $( COUNTER=0 $)


LET     WRITEC(CH) BE
        $( LET LINE = OUTPUT%BUFFER
           AND NEXT = OUTPUT%POINTER+1

           TEST  CH = '*N'
                 THEN $( LINE%0 := OUTPUT%POINTER
                         PACKSTRING (LINE,LINE)
                         (OUPUT%TRANSFER) (OUTPUT%FILE, LINE)
                         OUTPUT%POINTER := 0
                         COUNTER := COUNTER + 1
                      $)
                 OR
                      $( IF NEXT > BUFFERSIZE
                            $( WRITEC ('*N') ; NEXT := 1 $)
                         OUTPUT%POINTER := NEXT
                         LINE%NEXT := CH
                      $)
        $)
          .END OF EXAMPLE
```

Przykład

maczony programem J, co prowadzi do otrzymania programu użytkowego działającego na maszynie 1.

Schemat samoulepszającej się implementacji z użyciem języka BCPL zaczerpnięto z pracy [2].

LITERATURA

[1] Richards M., Whitby-Stevens C.: BCPL — The Language and its Compiler. Cambridge University Press, 1979

[2] Suffrin B. A.: A BCPL Implementation. University of Essex.

Samotesty

III/A. Podstawy sortowania zbiorów

Samotesty, opracowane i opublikowane przez Association for Computing Machinery, nie stanowią standardu egzaminacyjnego i są przeznaczone wyłącznie do samodzielnego sprawdzania własnej wiedzy [1]).

PYTANIA

1. Załóżmy, że mamy potasowaną talię 52 kart brydżowych. Która z poniższych manualnych metod będzie najszybsza dla posortowania tej talii, tak, aby trefle występowały przed karami, dalej kiery i piki, a w ramach kolorów — od dwójki do asa?

   a) rozdaj talię na cztery stosy według kolorów; każdy stos posortuj oddzielnie według wartości kart; złóż posortowane stosy według kolejności kolorów

   b) rozdaj talię kart wykładając karty na stół w przewidzianych na nie miejscach: złóż posortowane karty od dwójki treflowej do asa pikowego

   c) wybierz z talii wszystkie trefle; posortuj je oddzielnie; złóż posortowane karty od dwójki do asa na stos; postąp analogicznie z karami, kierami i pikami, składając posortowane karty na wierzch stosu treflowego

   d) rozdaj talię na 13 stosów według wartości kart; złóż stosy w kolejności tych wartości; rozdaj talię na cztery stosy według kolorów; złóż stosy w kolejności tych kolorów

2. Wiele komputerowych metod sortowania działa na zasadzie porównywania zawartości kluczy dwóch rekordów i zamiany tych rekordów miejscami, jeżeli nie występują w żądanym uporządkowaniu. Która z wymienionych poniżej metod tego rodzaju byłaby najszybsza?

   a) minimalizująca liczbę przestawień rekordów za cenę zwiększonej liczby porównań

   b) minimalizująca liczbę porównań poprzez wykorzystanie częściowych uporządkowań w zbiorze poddanym sortowaniu

   c) dostarczona przez producenta sprzętu

   d) do takiego oszacowania potrzebne są dodatkowe informacje o rozmiarach i organizacji pamięci operacyjnej, o liczności i uporządkowaniu zbioru podlegającego sortowaniu oraz o rozmiarach i charakterystyce pól kluczowych w sortowanych rekordach

3. Częściowe uporządkowanie danych w zbiorze, który ma być poddany sortowaniu, może mieć znaczny wpływ na szybkość tego procesu. Przy którym z niżej wymienionych uporządkowań pierwotnych sortowanie odbywać się będzie najszybciej?

   a) zbiór jest już w pełni uporządkowany

   b) zbiór, który ma być sortowany, jest uporządkowany akurat odwrotnie

   c) uporządkowanie zbioru, który ma być poddany sortowaniu, jest chaotyczne

   d) przy każdym z wymienionych wyżej uporządkowań początkowych sortowanie może trwać najkrócej, w zależności od obranej metody sortowania

4. Jakie jest minimum niezbędnych porównań kluczy rekordów, potrzebych do posortowania zbioru zawierającego sto rekordów?

   a) 0 c) 665 e) 4950
   b) 99 d) 2500 f) 5000

ROZWIĄZANIA:

ɐ — ƒ 'p — ɛ 'p — z 'p — ı

KOMENTARZE [2]):

ad 1. Knuth Donald E.: The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley, 1975; p. 170—171
Należy tu jednak zwrócić uwagę na fakt, że szybkość wykładania i zbieranie ze stolu kart jest uwarunkowana wprawą danej osoby i jej spostrzegawczością; zdolność ogarniania „układu ręki" jednym rzutem oka może spowodować, że w określonych przypadkach najszybszą mogłaby się okazać metoda „b".

ad 2. Martin William A.: Sorting. Computing Surveys, vol. 3, No. 4, December 1971, p. 147—174
Rich Robert P.: Internal Sorting Methods Illustrated with PL/1 Programs. Prentice-Hall, 1972

ad 3. Pewne programy sortujące, jak np. sortowanie metodą wartościowania (ang. ranking sort) działają najszybciej, gdy zbiór oryginalny jest w dużym stopniu uporządkowany. Inne jednak, jak np. sortowanie metodą powinowactwa (ang. ancestral sort) najlepiej działają, gdy rekordy występują w uporządkowaniu przypadkowym. Ale są też metody, jak np. metoda najmłodszej pozycji (ang. upward radix) w ogóle nie reagujące na sposób uporządkowania rekordów w zbiorze poddawanym sortowaniu.

ad 4. Sortowanie metodą „upward radix sort" omawia Knuth (str. 181), natomiast omówienie metody „address calculation sort" znajduje się u Martina (str. 169) — żaden z tych programów nie wymaga bezpośredniego porównywania kluczy dwu rekordów.

Adaptowali z angielskiego:
ADAM B. EMPACHER
LUDWIK J. ROSSOWSKI

[1]) Niniejszy samotest zamieszczony został w: Communictions of the ACM vol. 20, No. 9, September 1977, p. 621—624

[2]) Podane w przypisach pozycje są dość trudne do zdobycia — proponujemy przejrzeć książkę W. M. Turskiego: Struktury danych, WNT, 1980

Nowa rodzina IBM

W połowie listopada ub. r. towarzystwo handlu zagranicznego EXIMPOL zorganizowało w Warszawie seminaria promocyjne nowo wprowadzonego modelu 3 w rodzinie komputerów IBM-4361. Łącznie w seminariach wzięło udział ponad dwustu informatyków i dyrektorów krajowych przedsiębiorstw i instytucji. Imprezy te były wyraźnie ukierunkowane na dotychczasowych kilkunastu posiadaczy systemów IBM 370 i 360.

Z przedstawionych materiałów wynika, że firma IBM po ogłoszeniu we wrześniu 1984 zredukowanej skrajnie wersji zmodernizowanych modeli typu 4361 — sformułowała nową rodzinę komputerów. Rodzina ta odpowiada linii rozwojowej systemów 370, uprzednio zastępowanych komputerami rodziny 4331 oraz 4341. Architektura procesów rodziny 4361 została zaprojektowana przez ośrodek badawczy IBM w Boeblingen (RFN).

Aktualnie rodzina 4361 obejmuje trzy tzw. grupy modeli, oznaczane jako: 3, 4 i 5. Procesory dla wszystkich grup i modeli mieszczą się w standardowej obudowie o gabarytach 1,5×1,0×0,8 m. Przy oznaczeniu typu procesora numer grupy poprzedza symbol literowy określający pojemność pamięci operacyjnej: K — 2 MB, L — 4 MB, M — 8 MB, N — 16 MB. W grupie najniższej występują tylko dwa modele: K3 oraz L3. W średniej grupie jest możliwych pięć modeli: K4, L4, LK4, M4 i ML4. Znacznie większe pojemności pamięci występują w grupie najwyższej, ale ze względu na ograniczenia eksportowe trudno na razie oczekiwać, aby modele tej grupy mogły w najbliższej przyszłości uzyskać licencję eksportową do krajów RWPG, w sytuacji gdy w pozostałych grupach licencje można uzyskać obecnie tylko na modele najniższe, tj. K3 i K4, a i to jedynie w zakresie odpowiednio ograniczonych konfiguracji urządzeń dyskowych i teledacyjnych.

W dążeniu do wzrostu niezawodności działania sprzętu, oprócz rozbudowanego wewnętrznego procesora diagnostycznego, założono dla rodziny 4361 rozproszoną bazę danych diagnostycznych, z której oprócz ośrodka projektowego korzystają także zakłady produkcyjne i regionalne oddziały firmy. W wyniku tych przedsięwzięć oraz systematycznej kontroli profilaktycznej — jak utrzymywali przedstawiciele IBM z ośrodka wiedeńskiego — niezawodność rodziny 4361 ma być co najmniej 10-krotnie większa niż rodziny 370. **(A.B.E.)**

Naukowiec — biznesmenem

W firmie DIGITAL RESEARCH narodził się system operacyjny CP/M 80, który stał się międzynarodowym standardem. Czasopismo CHIP przeprowadziło z prezesem tej firmy GARY KILDALLEM wywiad na temat strategii na rynku oprogramowania dla mikrokomputerów 16-bitowych, który publikujemy poniżej. Czytelnikom INFORMATYKI warto przypomnieć, że rodzinę systemów CP/M omówiliśmy na naszych łamach w numerach 1—3/1983.

Red.: — Dzięki systemowi CP/M 80 udało się firmie DIGITAL RESEARCH stworzyć standard na rynku mikrokomputerów 8-bitowych. W jaki sposób osiągnięto ten niewątpliwy sukces?

Kildall: — W pierwszym okresie, po szybkim wprowadzeniu mikrokomputerów, istniały jedynie zalążki systemu operacyjnego, będące częścią interpretera języka BASIC. Sytuacja ta skłoniła mnie do zaprojektowania za pomocą opracowanego przeze mnie języka programowania PL/M systemu operacyjnego, który nazwałem CP/M (Control Program for Microcomputers). Początkowo produkt ten został zaoferowany firmie INTEL. Ponieważ oferta ta nie znalazła żadnego oddźwięku, postanowiłem wspólnie z Dorothy Mc Even stworzyć rynek zbytu przez odpowiednią działalność propagandowo-reklamową. Z chwilą gdy system osiągnął sukces w środowisku użytkowników-hobbystów, zrezygnowałem ze stanowiska profesora i kierownika katedry informatyki, zakładając w 1976 roku firmę DIGITAL RESEARCH. System CP/M stał się szlagierem, czego dowodem jest fakt jego użytkowania na ponad 350 tys. komputerach osobistych. Do systemu weszły następujące języki programowania: CBASIC, CIS COBOL, LEVEL II COBOL, PASCAL MT+ oraz PL/I.

Red.: — Jak wygląda dziś podaż systemów operacyjnych?

Kildall: — Podstawą oferty DIGITAL RESEARCH jest CP/M 80, który — jako system z dostępem jednego użytkownika — przeznaczony jest dla mikroprocesorów 8080, 8085 i Z80. Dla tego systemu 600 producentów oprogramowania opracowało ponad 3000 programów użytkowych. Oprócz CP/M 80 istnieje MP/M II, wielodostępny i wielozadaniowy system operacyjny czasu rzeczywistego. Może on obsłużyć do 16 stanowisk każdego z następujących urządzeń zewnętrznych: monitorów ekranowych, drukarek i dysków elastycznych o łącznej pojemności do 512 MB. Ofertę uzupełnia CP/Net, wspomagający znaczną liczbę protokołów transmisyjnych.

Red.: — Jakie systemy operacyjne oferuje Pan dla mikrokomputerów 16-bitowych?

Kildall: — Dla rodzin mikroprocesorów 8086 i 8088 oferujemy systemy CP/M 86, MP/M 86 oraz CONCURRENT CP/M 86. Dla mikroprocesorów Z8000 oraz MOTOROLA 68000 istnieją odrębne wersje. Za znaczące osiągnięcie uważam CONCURRENT CP/M 86. Pozwala on użytkownikowi, po odpowiedniej rozbudowie pamięci, realizować na komputerze osobistym wiele maszyn wirtualnych. Może przy tym przełączać się dowolnie pomiędzy wirtualnymi terminalami — w taki sposób, aby również programy z dużą intensywnością obliczeń, które wymagają okresowego dostępu, mogły przebiegać w obszarze drugoplanowym.

Red.: — Czy CP/M 80 stanowi dla firmy zamknięty rozdział, czy też przewiduje się jego dalszy rozwój?

Kildall: — Mikroprocesory 8-bitowe w bieżącym dziesięcioleciu potwierdzą swą pozycję na rynku. Dlatego oferujemy teraz CP/M PLUS. Najistotniejszymi jego cechami są możliwości obsługi błędów, funkcje pomocnicze oraz zakładanie plików wsadowych, pozwalające komputerowi pracować samodzielnie.

Red.: — Jakie stworzył Pan możliwości dla wymienności programów użytkowych?

Kildall: — Dla tych zadań stawiamy do dyspozycji programistów program zarządzania ekranem (Display — Manager) oraz zarządzania dostępem (Access-Manager). Stwarza to możliwość tworzenia konwersacyjnych monitorów ekranowych oraz zbiorów danych z dostępem wg klucza, bez dużych nakładów. Do tego dochodzą inne programy usługowe, ułatwiające programowanie, obsługę drukarek (tzw. spoling) oraz zarządzanie dyskietkami. Dla producentów oprogramowania szczególnie interesujący może być program tłumaczący XLT86, pozwalający przekształcać asembler 8080 na asembler 8086.

Red.: — Czy DIGITAL RESEARCH oferuje również uniwersalne narzędzia grafiki komputerowej?

Kildall: — Zaplanowano stworzenie całej biblioteki podprogramów plotera, które będzie można wywoływać za pomocą różnych języków programowania. W pierwszym etapie oferowany będzie pakiet do prezentacji dwuwymiarowej, który następnie zostanie rozszerzony o bibliotekę, za pomocą której będzie można generować grafikę wymaganą w dziedzinie zastosowań gospodarczych. Ostatnim produktem będzie pakiet grafiki dla zastosowań trójwymiarowych.

Red.: — Jaką rolę w Pana koncepcji odgrywa kompatybilność?

Kildall: — Jest to punkt, który niekiedy jest nam stawiany jako zarzut, a jest powszechnie odbierany jako wada oprogramowania. Podczas opracowania wersji CP/M dla mikroprocesorów 16-bitowych zwracaliśmy uwagę na to, aby sprzęgi użytkownika i zarządzanie dyskietkami pozostały bez zmian. Oznacza to, że zbiory w kodzie ASCII, zapisane pod nadzorem CP/M 80, mogą być odczytane również pod nadzorem CP/M 86 i odwrotnie. Wysiłki, aby utrzymać wymienność zbiorów i rozkazów, przewijają się przez całą gamę naszych produktów. Podstawą dla nas jest CP/M 80, ponieważ rodowód wielu naszych użytkowników wywodzi się z tego poziomu rozwoju systemu. Nadbudowując tę koncepcję stworzyliśmy w naszych produktach coraz więcej możliwości efektywnego wykorzystania mikrokomputerów. Pierwszym krokiem była integracja wielu użytkowników, drugim — możliwość korzystania i budowa sieci komputerowych.

Red.: — Jak ocenia Pan waszą pozycję rynkową?

Kildall: — Głównymi konkurentami były firmy APPLE i RADIO SHACK, które opracowały własne systemy operacyjne. Tendencje do stosowania CP/M były jednak tak silne, zwłaszcza dzięki bardzo szerokiej gamie dostępnych języków, że wskutek zapewnienia serwisu i szkolenia użytkowników utrzymaliśmy dotychczasową pozycję rynkową. Przyczynia się do tego również fakt, że nowe opracowania są zgodne z produktami już istniejącymi. Aby utrzymać i wzmocnić pozycję rynkową, oferowane są liczne efektywne narzędzia programowania. Rynkiem rządzi bowiem bardzo proste prawo: **kto oferuje najwięcej oprogramowania, ten osiągnie największy sukces.** Jesteśmy w trakcie rozbudowy sieci serwisowej dla Europy z centralą w Wielkiej Brytanii i oddziałami w RFN i Francji. Nasze produkty związane z konwersacją ekranową oraz podręczniki, z którymi styka się użytkownik, zostaną przetłumaczone na języki odpowiednich krajów.

Opracował **W.K.**

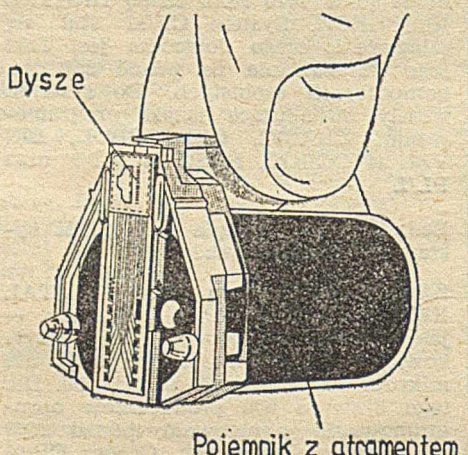
Drukarka strumieniowa HP

Drukarka HP Thinkjet model 2225 A/B/C. Tłumaczę to jako **strumieniowa**, gdyż elementem piszącym jest tu strumień atramentu wyrzucany ze specjalnej dyszy. Powie ktoś, że tego typu drukarki są znane już od dawna — są wprawdzie ciche i szybkie, ale też — bardzo drogie; a tu jeszcze HEWLETT-PACKARD! Tak, HP kojarzy się czasem z Higher Prices, Wyższe Ceny...

To jednak przeszłość. Omawiany model przeznaczony jest do obsługi komputerów osobistych i to w pełnym tego słowa znaczeniu. Przede wszystkim cena: tylko 495 dol., co w praktyce oznacza, że będzie ją można kupić już za ok. 350 dol. plus podatek. Wymiary drukarki są bardzo małe: 30×20×9 cm³, przy czym model B — wyposażony w sprzęg HP-IL — zasilany jest z baterii. A więc w pełni przenośna drukarka strumieniowa!

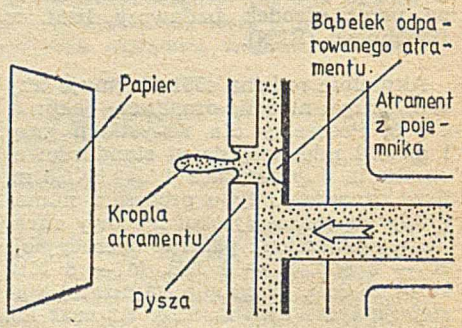
Modele A i C wyposażone są odpowiednio w HP-IB oraz w sprzęg Centronics i zasilane są z sieci. Drukarka waży mniej niż 3,5 kg i podczas pracy generuje hałas o natężeniu 50 db, co odpowiada mniej więcej niezbyt głośnej rozmowie. Głowica drukująca jest wymienna i wystarcza do zapisania ok. 500 arkuszy formatu A4. Cena głowicy, która jest zarazem jednorazowym pojemnikiem na atrament, wynosi 7,95 dol. Istota działania drukarki zasadza się na impulsowym ogrzewaniu (do wrzenia) atramentu, co powoduje wyrzucanie strumienia tego płynu przez dyszę. Żywotność drukarki określono, na 3,5 miliona linii, czyli ok. sto tysięcy arkuszy formatu A4. Z moich obliczeń wynika, że oznacza to w praktyce konieczność wymiany głowicy piszącej raz — dwa razy w roku i możność używania drukarki do końca życia!

HP przewiduje użycie specjalnego, powlekanego papieru (w USA ok. trzy razy droższy od normalnego), ale testy wykazały, że nawet na normalnym papierze uzyskuje się dostateczną kontrastowość wydruków. Precyzja pisania jest duża: standardowo Thinkjet pisze jak zwykła drukarka mozaikowa z głowicą 11×12 punktów. W trybie graficznym (kopiowanie grafiki z ekranu monitora) można używać głowicy 96×96 punktów na cal kwadratowy (ok. 38×38 punktów/cm²) lub podwyższonej rozdzielczości 192×96 punktów/cal² (77×38 punktów/cm²). Przewidziano cztery rodzaje czcionki: normalną (12 znaków/cal), rozszerzoną (6 znaków/cal), zwężoną (21 znaków/cal) oraz rozszerzoną-zwężoną (10 znaków/cal).



Odpowiadają one 80, 40, 142 lub 71 znakom w wierszu. Można drukować sześć lub osiem wierszy na cal (odstęp międzywierszowy 2,1 lub 1,6 mm). Średnia prędkość drukowania wynosi 160 znaków/s, przy czym drukowanie jest dwukierunkowe.

Co do rodzaju liter, to oprócz standardowego alfabetu łacińskiego przewidziano też narodowe znaki języków: duńskiego, holenderskiego, angielskiego, fińskiego, francuskiego, niemieckiego, szwedzkiego, włoskiego, norweskiego, portugalskiego, hiszpańskiego. Nie przewidziano dotąd znaków diakrytycznych polskich.



Niestety, wydaje się, że HP jest firmą tradycyjną i niechętnie wprowadza wszelkie nowości; dlatego też wybór rodzaju i kroju pisma dokonuje się w Thinkjet tylko programowo, gdy tymczasem konkurencja (EPSON!) produkuje już modele sterowane też bezpośrednio przez użytkownika (tzw. Finger Print). W przypadku modelu przenośnego, zapewniającego łatwą możliwość przyłączenia drukarki do różnych komputerów, a tym samym używania jej do pisania różnych tekstów, konieczność każdorazowego konfigurowania drukarki przez program stanowi pewną niewygodę.

Wszystko wskazuje na to iż era tanich, uniwersalnych drukarek strumieniowych już się zaczęła. Przedwczesne były — jak się wydaje — głosy o zdecydowanej dominacji IBM na rynku komputerów osobistych — jeszcze ciągle jest wielu klientów, którzy cenią sobie nowoczesne, choć nieco droższe wyroby. Thinkjet stanowi właśnie ofertę dla nich.

Oprac. J.T.

CSK—Komputer Studio Kajkowscy

81-505 GDYNIA ORŁOWO ul. Balladyny 3B, tel. 29-00-18

Komputer osobisty może być przydatny niemal na każdym stanowisku pracy. Wymaga jednak odpowiedniego oprogramowania użytkowego. W ramach tego oprogramowania oferujemy zainteresowanym dostawę uniwersalnych pakietów programowych:

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

To doskonałe narzędzia pracy dla każdego. Aby z nich korzystać, nie trzeba być informatykiem! Zupełnie samodzielnie można tworzyć złożone systemy zarządzania przedsiębiorstwem, każdym przedsiębiorstwem; nawet najbardziej specyficzne uwarunkowania nie są przeszkodą.
To jednak jeszcze nie wszystko... Kiedy dotychczasowe problemy łatwo i szybko zostały rozwiązane — pojawiają się zupełnie nowe. Można wtedy bez kłopotów samemu udoskonalić dotychczasowy system!

BANK DANYCH CSK, TABPLAN CSK, TEKST CSK, TRANSCOM CSK

składają się w zakładowe systemy płacowe, osobowe, finansowo-księgowe lub magazynowe. Korzystając z nich, z łatwością można prowadzić planowanie, kalkulacje i sprawozdawczość. Można też sporządzać kosztorysy i oferty, a nawet prowadzić „automatyczną" korespondencję czy redagować dowolne teksty. Można wreszcie skorzystać z już zgromadzonych zasobów na komputerze ODRA (pod nadzorem systemu GEORGE-3), wykorzystując komputer osobisty jako inteligentny terminal — stację lub emulator TTY.

Nowość:

Oferujemy system operacyjny kompatybilny z CP/M 2.2. dla mikrokomputerów ROBOTRON 5120/5130 oraz systemy finansowo-księgowe FK dla dowolnych mikrokomputerów

---

Szczegółowych informacji udziela:

CSK—Komputer Studio Kajkowscy

81-505 GDYNIA ORŁOWO, ul. Balladyny 3B, tel. 29-00-18

EO/129/K/85

Sztuczna inteligencja i systemy ekspertowe

Kongres IFIP, który odbył się w Paryżu na początku października 1983, wywołał kolejną falę zainteresowania sztuczną inteligencją, a dokładniej — systemami ekspertowymi. Dziedzina ta, będąca jeszcze na marginesie tradycyjnej informatyki, jest w stanie zaproponować oryginalne rozwiązania problemów rozwiązywanych często niewłaściwie lub w ogóle dotąd nie rozstrzygniętych. O jej znaczeniu niech jednak świadczy fakt, że stanowić ona będzie serce japońskich komputerów Piątej Generacji.

Sztuczną inteligencję można zdefiniować jako dziedzinę informatyki, której przedmiotem badań jest działalność intelektualna człowieka w tych przypadkach, w których nie jest znane żadne proste rozwiązanie algorytmiczne. Ta definicja, na pewno niekompletna i z trudem broniąca się przed krytyką, pozwoli przedstawić podstawowe problemy, które napotykają twórcy omawianej dziedziny informatyki.

★

Zajmijmy się — na przykład — rozumieniem języka. W tym przypadku należy dysponować zbiorem zawierającym słownik języka, jego gramatykę (z listą wyrażeń potocznych i często popełnianych błędów składni) oraz informacje dotyczące semantyki, niezbędne do dokonania analizy zdań. Zbiór ten zawiera tak wiele informacji natury symbolicznej, że wyłania się problem **sposobu reprezentowania i posługiwania się tą wiedzą.**

Spójrzmy choćby na krótki fragment zdania: „kapitan statku, który płynie". Można dostrzec z jak ogromnej ilości informacji składniowych, semantycznych i pragmatycznych trzeba skorzystać, aby osiągnąć cel, którym jest poprawne rozumienie języka.

Analogiczny problem występuje także w przypadku systemów ekspertowych, tzn. systemów ukierunkowanych na symulację postępowania eksperta rozwiązującego problem z zakresu jednoznacznie zdefiniowanej specjalności [1]).

★

Następną cechą charakterystyczną omawianej klasy zagadnień jest brak algorytmu prowadzącego w rozsądnie krótkim czasie do rozwiązania. Oznacza to, że nie można z góry zdefiniować etapów prowadzących do rozwiązania. Na przykład — nie istnieje algorytm pozwalający tłumaczyć tekst z jednego języka na inny.

W takich przypadkach należy zdefiniować metody lub **heurystyki**, które zapewnią rozsądny kompromis między jakością rozwiązania a czasem obliczeń niezbędnych do jego uzyskania. Rozwiązania nie będą jednak optymalne, nawet wtedy, gdy można wykazać ich istnienie w sposób teoretyczny.

Jeśli więc nawet wiadomo o istnieniu strategii pozwalającej osiągnąć w najgorszym przypadku „pat", to nie jest możliwe analizowanie na każdym etapie poszukiwania rozwiązania, wszystkich możliwych posunięć, prowadzących do wyboru najlepszego (czas obliczeń byłby astronomiczny). Praktyczniej jest więc wybierać najlepszy krok w oparciu o pewną funkcję oceny, którą wyznacza się w większym lub mniejszym stopniu empirycznie i która tworzy heurystykę.

★

Nie tylko nie ma algorytmu automatycznie tłumaczącego teksty z jednego języka na inny, ale nawet nie istnieje tłumaczenie, o którym można by powiedzieć, że jest dokładne. Powyższe stwierdzenie rzuca światło na trzecią charakterystyczną cechę, wspólną dla większości problemów rozwiązywanych metodami sztucznej inteligencji: **przybliżony charakter wiedzy**, czy raczej — reguł, z których się korzysta.

★

Jeżeli metodami sztucznej inteligencji udaje się przetwarzać (często sposobami niealgorytmicznymi) dane typu symbolicznego, implikuje to z jednej strony możliwość przyswajania wiadomości i łączenia ich w struktury (działania koncepcyjne!), a z drugiej strony — zdolność rozumowania. Ponieważ zwykłe języki algorytmiczne okazały się mało przydatne, stworzono inne — **języki sztucznej inteligencji** — spełniające nowe wymagania, a mianowicie:

- LISP, stworzony przez J. Mc Carthyego (MIT) w 1956 r. do opisu i działań na listach — używany najczęściej
- PROLOG, rozwinięty przez A. Colmerauera (Uniwersytet w Marsylii) w 1974 r. oparty na logice predykatów — główny reprezentant języków „deklaracyjnych"
- SMALLTALK, język najnowszy, stanowiący wynik prac C. Hewita i A. Kaya; jest to przykład tzw. języka działającego w sposób aktywny (pisanie programów w takim języku sprowadza się do definiowania klasy obiektów, ich cech i sposobów, w jaki te obiekty mogą porozumiewać się za pomocą tzw. informacji).

Większość programów sztucznej inteligencji realizowana jest obecnie za pomocą dwóch pierwszych języków.

★

Systemy ekspertowe wykorzystują zbiór reguł empirycznych, a często również aproksymacji rządzących określoną dziedziną wiedzy. Wśród różnych tematów, które obejmują tego rodzaju systemy na pierwszy plan wysuwają się: diagnostyka medyczna (Mycin), poszukiwania górnicze (Prospector), budowanie systemów informatycznych (R1), rozwiązywanie problemów chemicznych (Meta-Dendral), itp.

Oprócz reguł empirycznych systemy te wykorzystują także fakty związane z opracowywanym problemem, z których można wyciągnąć wnioski przydatne do rozwiązania postawionego problemu.

Cechą wyróżniającą systemy ekspertów jest to, że usiłuje się maksymalnie rozłączyć reguły rozumowania (baza reguł), dane (baza danych) i sposób posługiwania się tą wiedzą (system kontroli). Jeżeli nawet struktura taka jest mniej skuteczna niż program klasyczny, to na pewno przewyższy go budową modułową i łatwością testowania, co w tym przypadku jest sprawą zasadniczą. Dzięki tym zaletom, nawet jeśli system zablokuje się ze względu na brak potrzebnej reguły lub złe jej sformułowanie, to wystarczy dołączyć ją do bazy reguł, bez konieczności zmiany reszty systemu. Klasyczny program nie może być zaadaptowany w tak prosty sposób.

Wyobraźmy sobie uproszczony system zawierający następujące dwie reguły:

JEŻELI pokryty piórami I jajorodny TO ptak
JEŻELI ptak TO lata

Jeśli do systemu wczytane są dane „pokryty piórami" i „jajorodny", to wysuwa on wniosek, że chodzi o ptaka — istotę latającą. Jeżeli jednak pomysłodawca zauważy, że struś jest przypadkiem wyłamującym się z reguły pierwszej, wystarczy zastąpić regułę drugą przez nowe sformułowanie

JEŻELI ptak I NIE struś TO lata.

Pozostaje tylko mieć nadzieję, że w bliższej lub dalszej przyszłości systemy (bardzo) inteligentne będą mogły, poprzez autoanalizę własnej wiedzy, same określać reguły, które nie zostały uwzględnione przez projektanta systemu: w przypadku naszkicowanej powyżej struktury automatyczna modyfikacja wiedzy (uczenie się) przebiegnie łatwo.

★

Systemy ekspertowe muszą mieć — co oczywiste — zapewniony dostęp do ogromnych ilości informacji (wyobraźmy sobie zbiór informacji niezbędnych do postawienia diagnozy na podstawie prześwietlenia płuc). Elementy te będą zgromadzone **w bazie danych**, do której dostęp zarówno dla programów jak i użytkowników powinien być znacznie prostszy i bardziej „inteligentny" niż obecnie. W celu zwiększenia efektywności tradycyjnych systemów zarządzania baz danych, należy je tak

[1]) Wzorcowy system ekspertowy MYCIN został opisany w artykule Andrzeja Szałasa w INFORMATYCE nr 9/1984 — przyp. red.

zmodyfikować, aby były zdolne wykorzystywać wiadomości o typach danych, którymi manipulują.

Na przykład — w bazie danych zawierającej relację zależności w pracy:

PRACOWNIK	ZWIERZCHNIK
Jan	Hubert
Paweł	Hubert
Andrzej	Edward

system powinien m.in., odpowiedzieć na problem sformułowany następująco:

Wybierz (PRACOWNIK) zależny od ZWIERZCHNIK gdy ZWIERZCHNIK = „Hubert"

★

Dostęp do baz danych powinien być realizowany w języku quasi-naturalnym, tj. w języku potocznym użytym w ograniczonym zakresie. Zrozumienie języka potocznego i rozpoznanie jego form, uwzględnienie czynników **percepcji** niezbędnych do zrealizowania prostego dostępu do inteligentnych systemów informacji musi opierać się na tzw. reprezentacji wiedzy. W tym momencie do problemów fizycznych (wybór czujnika, przetwarzanie i kompresja sygnałów, kodowanie itp.), które napotykają twórcy sztucznej inteligencji, dołączają się zagadnienia związane z koniecznością zdefiniowania tego, co jest wiedzą i określenia sposobu, w jaki powinna być ona reprezentowana, a więc zagadnienia o podstawowym znaczeniu we wszystkich dziedzinach sztucznej inteligencji.

★

Metody analizy otoczenia są podstawą systemów zrobotyzowanych. Mimo że jedynymi dotychczasowymi realizacjami **robotyki** (techniki z pogranicza informatyki, elektroniki i mechaniki) są automaty programowane, ewolucja w kierunku robotów inteligentnych jest jednak nieuchronna[2]). Wiąże się z tym konieczność uwzględnienia hierarchicznych poziomów rozumienia otaczającego świata.

★

Jednym z tematów stanowiących siłę napędową rozwoju sztucznej inteligencji jest **dowodzenie twierdzeń** — dziedzina bez wątpienia bardziej szlachetna niż systemy ekspertowe, które operują danymi empirycznymi i przybliżonymi. Jako jej narzędzia rozwinęły się techniki dedukcji, język zdań, logika predykatów — podstawy metod reprezentacji i operowania wiedzą.

★

W przypadku **programowania automatycznego** punktem wyjścia dla systemowej realizacji programu są tylko specyfikacje formalne. W ramach prac nad takim programowaniem rozwiązano częściowo problemy semantyki, dzięki tzw. semantyce aksjomatycznej, denotacyjnej itp.

★

Ten krótki przegląd dziedzin sztucznej inteligencji dobrze uwidacznia, że jednym z podstawowych problemów jest sposób reprezentowania i operowania wiedzą. INFORMATYKA będzie się starała kontynuować wywołany temat.

Opracowała HANNA KONTKIEWICZ
na podstawie MINIS et MICROS
z 27 lutego 1984

[2]) Por. cykl artykułów Teresy Wilczek i Cezarego Zielińskiego o robotyce — INFORMATYKA nr 5, 7, 8, 9/1984 — przyp. red.

KONFERENCJE

Centrum Obliczeniowe Politechniki Wrocławskiej organizuje w dniach 22—23 października 1985 konferencję naukową pn.:

„Sieci komputerowe — usługi, protokoły, modele".

Jej celem jest przedstawienie osiągnięć w zakresie projektowania i badań sieci komputerowych w Polsce.

W czasie trwania konferencji zostaną zaprezentowane referaty i komunikaty dotyczące następujących tematów:

- Modele rozległych sieci komputerowych (typu WAN)
- Protokoły w sieciach komputerowych i ich standaryzacja (protokół: sieciowy, transportowy, sesji, prezentacji)
- Bazy danych w sieciach komputerowych
- Modele i usługi sieci lokalnych (typu LAN).

Przewidziany jest także pokaz działania Międzyuczelnianej Sieci Komputerowej (MSK) i sprzętu.

Zainteresowani udziałem w konferencji powinni przesłać w terminie do dnia 1 kwietnia br. zgłoszenie oraz streszczenie komunikatu (1—2 str. maszynopisu). Opłata za udział w konferencji wynosi 3800 zł, które należy przesłać także w terminie do 1 kwietnia br. pod adresem: **Centrum Obliczeniowe Politechniki Wrocławskiej, Pl. Grunwaldzki 9, 50-372 Wrocław, tel. 20-34-31.**

Konto: Politechnika Wrocławska, NBP V O/M Wrocław 93057-3418 (z zaznaczeniem: Centr. Oblicz. „Sieci komputerowe").

Pozorne złagodzenia

Oczekiwana od dłuższego czasu zmiana dotychczasowych zasad eksportu sprzętu informatycznego produkcji zachodniej do krajów RWPG, zasad sformułowanych jeszcze w 1976 roku, wywołała zrozumiałe poruszenie w światku informatycznym. Z jednej bowiem strony — według nowych zasad zwiększono prawie 3,5-krotnie dopuszczalną pojemność pamięci wewnętrznych i ponad 2-krotnie pojemność pamięci zewnętrznych, a także podwyższono granicę szybkości przesyłowej kanałów (tabela). Z drugiej strony — pojawiły się wyraźne obostrzenia jakościowe: z zezwoleń eksportowych wyłączono całkowicie komputery sieciowe, sprzęt transmisji danych o szybkości ponad 9600 bodów oraz systemy graficznego wspomagania prac projektowych.

Nowe zasady przewidują dla niektórych grup sprzętu informatycznego możliwość udzielania okresowych lub bezterminowych uproszczeń trybu otrzymywania licencji eksportowej, przez skrócenie drogi obiegu dokumentów. Pełny cykl obejmował bowiem:

- przedłożenie wniosku odpowiedniemu urzędowi państwowemu w kraju macierzystym eksportera
- skierowanie wniosku do komitetu COCOM (Committee for COMECON Countries) w Paryżu
- przekazanie kopii wniosku każdej z 15 delegacji członkowskich (USA, Europa Zachodnia, Japonia)
- uzyskanie opinii macierzystych organizacji rządowych dla każdej delegacji z osobna
- często spotykane odraczanie terminu plenarnego rozpatrzenia wniosku, na wniosek niektórych delegacji, które nie zdążyły przygotować opinii
- powiadomienie właściwego urzędu państwowego w kraju macierzystym eksportera
- powiadomienie eksportera o przyznaniu lub odmowie udzielenia licencji eksportowej.

Procedura taka oznaczała w praktyce nawet większą uciążliwość dla krajów zachodnich aniżeli dla krajów RWPG, których udział w światowym handlu komputerowym mierzy się zaledwie na poziomie ok. 1%, jako że ograniczenia COCOM dotyczą także obrotów handlowych między krajami członkowskimi. Liberalizacji procedury można oczekiwać przede wszystkim w zakresie komputerów osobistych — byle nie łączonych w zbyt duże sieci (praktycznie ponad 20-jednostkowe).

Na wprowadzeniu nowych zasad COCOM skorzystają przede wszystkim użytkownicy mocno już wyeksploatowanych komputerów zachodnich, oczywiście — zdolni finansowo do zakupu nowocześniejszego sprzętu. Wiele bowiem nowoanonsowanych rodzin komputerów oferuje już jako podstawową wielkość pamięci operacyjnej — 2 MB, wykraczającą poza dawną barierę 0,75 MB, praktycznie bezsensowną przy obecnej masowej produkcji pamięci półprzewodnikowych, ale skutecznie blokującą wszelkie próby modernizacji posiadanego sprzętu.

Natomiast po wprowadzeniu nowych przepisów niewątpliwie stracą wszyscy ci, którzy liczyli na przechwycenie zachodniej technologii graficznego wspomagania prac projektowo-inżynierskich, a w związku z tym nie podjęli własnych prac badawczo-rozwojowych w tym zakresie. W tej chwili bowiem systemy CAD/CAM zostały uznane za podstawę suwerenności gospodarczej krajów rozwijających takie systemy.

Oczywiście, byłoby dziwnym oczekiwać, że kraje NATO zrezygnują nagle z ogólnych ograniczeń, zapewniających przynajmniej teoretycznie rozwieranie się nożyc postępu informatycznego w najistotniejszych strategicznie dziedzinach. Nie ma się więc co zachwycać liberalizacją ograniczeń w dziedzinie klasycznych zastosowań informatyki, raczej — co prędzej wykorzystać nowe zasady importu.

ADAM B. EMPACHER

KONFERENCJE

MIKROKOMPUTER'84

- W dniach 24—27 września 1985 roku w Bierutowicach koło Karpacza w DW „Szczyt", ul. Śnieżki 6 odbędzie się I Międzynarodowa Szkoła nt.: MIKROKOMPUTERY — Projektowanie — Praktyka — Nauczanie „MIKROKOMPUTER '85". Szkoła organizowana jest przez Instytut Cybernetyki Technicznej Politechniki Wrocławskiej.
- CEL TEGOROCZNEJ SZKOŁY: Wymiana doświadczeń z zakresu kształcenia w problematyce techniki mikroprocesorowej w szkołach wyższych oraz w ramach szkoleń specjalistów z przemysłu (studia podyplomowe, doskonalenie kadr inżynierskich). Ponadto, prezentacja najnowszego dorobku naukowo-badawczego w zakresie techniki mikroprocesorowej.
- TEMATYKA: Przewiduje się następujące sekcje:
  — projektowanie i zastosowanie systemów mikroprocesorowych w dydaktyce
  — podstawowa — technika mikroprocesorowa
  — nauczanie techniki mikroprocesorowej
  — organizacja laboratoriów dydaktycznych.
- JĘZYK SZKOŁY: angielski, rosyjski, polski
- UCZESTNICTWO: Zgłoszenia udziału należy dokonać w terminie do dnia 1.07.1985 r.
- KOSZT UCZESTNICTWA: Wraz z przesłaniem zgłoszenia należy przekazać na konto: NBP V OM Wrocław, nr 93057-3418-131 opłatę w wysokości 8700 zł.
  W zakres kosztów uczestnictwa wchodzi udział w obradach, otrzymanie materiałów, wyżywienie i zakwaterowanie.
  Wpłaty powinny zawierać imię i nazwisko osoby za którą dokonywana jest wpłata oraz skrót IGT — MIKROKOMPUTER '85.
- WYSTAWA: Przewiduje się zorganizowanie wystawy sprzętu i oprogramowania mikrokomputerów przez zainteresowanych użytkowników.
- KOMITET ORGANIZACYJNY: Doc. dr hab. Wojciech Zamojski (przewodniczący), dr inż. Maria Chałon, dr inż. Ryszard Jacewicz, dr inż. Ireneusz Jóźwiak.
- ADRES DO KORESPONDENCJI: „MIKROKOMPUTER '85"
  Instytut Cybernetyki Technicznej Politechniki Wrocławskiej
  ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland
  tel.: 20-36-94; 22-80-77; 20-32-88; 20-28-23. teleks: 07 12254 pwr pl; 07 12559 pwr pl.

Recenzje

Programowanie operacji wejścia-wyjścia

Autorzy projektów nowych języków programowania, skupieni na opracowywaniu efektywnych, strukturalnych konstrukcji języka, mało uwagi przykładali do zaprojektowania wygodnych mechanizmów specyfikacji operacji wejścia-wyjścia. Problem ten występuje w takich popularnych językach jak FORTRAN, PL/I, ALGOL-60 czy PASCAL. Dodatkowym utrudnieniem w opracowaniu prostych rozwiązań standardowych okazywała się konieczność uwzględnienia właściwości systemów cyfrowych, w których mają być wykonywane programy. W rezultacie dołączone do języków wysokiego poziomu operacje wejścia-wyjścia są w znacznym stopniu obciążone cechami aktualnie wtedy istniejących systemów (np. dla języka PL-I — systemu OS maszyn IBM) lub też pozostawione są do zaprojektowania implementatorom języka (np. ALGOL-60). Wynika stąd problem przenoszenia programów zapisanych co prawda w standardzie języka, ale wykorzystywanych w różnych systemach. Z reguły bowiem wtedy największe różnice występują w realizacji operacji wejścia-wyjścia.

Pochodną takiego stanu rzeczy są trudności programistów w poprawnym zastosowaniu efektywnie działających operacji wejścia-wyjścia w projektowanych przez nich programach. W początkowym okresie poznawania i stosowania nowego języka programowania, wykorzystywane są najprostsze konstrukcje operacji wczytywania danych i wyprowadzania wyników (dla początkujących najwygodniejsze są, ze względu na standardowe formaty wprowadzania i wyprowadzania wartości, instrukcje INPUT i PRINT z języka BASIC). Niestety, wraz ze wzrastającymi potrzebami programu, intensywnie wykorzystującymi środowisko systemu do przechowywania złożonych zbiorów danych, programiści napotykają wiele barier i niejasności. Ponadto firmowe podręczniki opisujące język i jego implementację (Reference Manual), a także podręczniki typu przewodników (Guide) czy inne tego typu publikacje — w małym tylko stopniu zwracają uwagę i nie wyjaśniają dogłębnie oprogramowania operacji wejścia-wyjścia.

Dlatego też z dużą uwagą należy odnotować pojawienie się na naszym rynku wydawniczym zbioru pozycji zajmujących się programowaniem operacji wejścia-wyjścia w najpopularniejszych językach wysokiego poziomu — ALGOLU-60, FORTRANIE, PASCALU, PL/I ADZIE. Autor, Jan Bielecki, w wydanym przez Politechnikę Warszawską zbiorze skryptów [1, 2, 3, 5, 6] zajął się przekazaniem podstawowych informacji i — co ważniejsze — idei, przybliżających programistom tę tematykę.

W ALGOLU-60 czynności obsługi operacji wejścia-wyjścia nie mogą być w naturalny sposób wyrażone za pomocą instrukcji języka. W pracy [1] omówiono więc standardowy zestaw procedur pierwotnych i rozszerzający ich możliwości zestaw procedur podstawowych, umożliwiających oprogramowanie operacji wejścia-wyjścia. Prezentując opracowanie Międzynarodowej Organizacji Normalizacyjnej (ISO), autor przedstawił specyfikacje tych procedur oraz omówił ich czynności. Szczególna uwaga została zwrócona na wyjaśnienie zasad wykorzystywania wzorców determinujących postać wyprowadzanych przez program wartości. Interesujące jest wyjaśnienie interpretacji wzorców w stosunku do cech opisujących strony drukarki wierszowej. Opanowanie tych zagadnień przez programistę jest istotnym problemem w opracowywaniu algorytmu programu. Autor nie ogranicza się do omówienia rozwiązań standardowych, lecz podaje także efekty działań tych procedur w implementacji języka ALGOL-6000 na komputerze CYBER 72.

FORTRAN zawiera istrukcje wprowadzania i wyprowadzania informacji oraz pomocnicze instrukcje wejścia-wyjścia. Opierając się na normie języka FORTRAN, autor w drugiej części skryptu [1] przedstawia wyczerpująco te zagadnienia. Pełne opisy instrukcji są zilustrowane szeregiem przykładów prezentujących możliwości zastosowań tych konstrukcji w praktyce. Wychodząc ze słusznego założenia, że najlepiej uczymy się na błędach (ileż to już razy dopiero wielokrotne próby uruchomienia i testowania programu wykazały nam, że popełniliśmy błąd — niemal trywialny!), autor przedstawia też przykłady błędnie zastosowanych konstrukcji. A trzeba tu dodać, że na pierwszy rzut oka wyglądają one całkiem poprawnie. Dodatkową ilustracją, da-

jącą dużo do myślenia na temat rzetelności implementacji kompilatorów języka FORTRAN, są przykłady kompilatorów FTN (CYBER 72), XFAM (ICL-1900) i IEWL (IBM 360/370 poziom G).

Uzupełnieniem wiadomości o programowaniu wejścia-wyjścia w języku FORTRAN IV/IBM poziomu G jest omówienie współpracy z plikami o organizacji bezpośredniej, rozszerzającej zbiór instrukcji języka wzorcowego.

Skrypt [1] zawiera też poszukiwane przez wielu programistów omówienie zasad metod współpracy z ploterem (model bębnowy typu 563 z konfiguracji CYBER 72). Przedstawiono bibliotekę procedur standardowych systemu CALCOMP, wykorzystywanych przez kompilator FTN języka FORTRAN, z wyczerpującymi opisami. Następnie, dla ilustracji, zamieszczono gotowy do wykorzystania program do wykreślania linii gładkich oraz program do wykonywania wykresów trójwymiarowych (szkoda tylko, że obecnie CYBER 72 znajduje się w coraz gorszej kondycji). Autor twierdzi jednak, że przeniesienie tego oprogramowania na inne konfiguracje nie powinno przysporzyć wielu problemów.

PASCAL — na tle dość skomplikowanych metod programowania operacji wejścia-wyjścia w innych językach — ma stosunkowo proste zasady, sprowadzające się w zasadzie do procedur PUT, GET, RESET i REWRITE. W skrypcie [1] omówiono te procedury oraz ich działanie na plikach rekordowych i znakowych, a także tekstowych.

Przedstawienie programowania operacji wejścia-wyjścia w języku PL/I jest treścią drugiego skryptu [2]. Książka ta jest kompletna w tym zakresie; wymaga jedynie znajomości podstaw programowania w języku PL/I (co jest zrozumiałe) oraz wiadomości na temat systemu OS/360/370-RIAD. Prezentację materiału oparto na procesorze języka PL/I F oraz na procesorze rozszerzonego języka PL/I, zwanym procesorem optymalizacyjnym. Omówiono tu dokładnie współpracę z plikami strumieniowymi (z atrybutem STREAM) oraz plikami rekordowymi (z atrybutem RECORD). Wszystkie opisane konstrukcje języka są zilustrowane przykładami prezentującymi możliwości ich zastosowań. Naturalne powiązanie języka PL/I z systemem OS wiąże się z koniecznością omówienia niektórych dyrektyw systemowych opisujących wykonywanie zadań. Zagadnienia te znajdują się również w tej książce, a wyczerpujące w treści dodatki dostarczają informacji między innymi o atrybucie ENVIRONMENT, sytuacjach wyjątkowych we współpracy z plikami oraz niektórych programach systemowych.

Skrypt [3] przedstawia system OS, wykorzystując dla ilustracji wykonywanie programów w PL/I. Tak więc obie książki [2, 3] wzajemnie się uzupełniają, naświetlając zasady oprogramowania operacji wejścia-wyjścia od strony języka oraz z punktu wymagań systemowych. W zastosowaniach praktycznym elementem tych prac są wzory służące do projektowania wielkości i struktury zbiorów.

Odrębnym opracowaniem na temat programowania operacji wejścia-wyjścia jest skrypt poświęcony programowaniu w środowisku wirtualnym [5]. Zagadnienie to zostało przedstawione na przykładzie systemu VSAM (Virtual Storage Access Method), znanego (w terminologii IBM) jako metoda dostępu VSAM. Po dokładnym omówieniu złożonych organizacji zbiorów danych i przedstawieniu zasad posługiwania się dla ich przetwarzania językiem PL/I, autor dokonuje interesującego porównania metod dostępu ISAM i VSAM.

Tak więc po uzupełnieniu tych prac pozycją [4] otrzymaliśmy kompletny zestaw opisu zasad i metod projektowania dowolnie złożonych operacji wejścia-wyjścia z wykorzystaniem języka PL/I w środowisku systemu OS. I chociaż język ten nie jest najnowocześniejszy, to jednak — ze względu na liczne zastosowania — długo jeszcze będzie w użytkowaniu.

Język ADA nie ma w Polsce dostępnych kompilatorów. Stąd jego wykorzystanie jest na razie ograniczone. Należy jednak sądzić, że w przyszłości stanie się on językiem powszechnego użytku. W przygotowanej do druku pracy [6] zaprezentowany jest standard operacji wejścia-wyjścia, zdefiniowany w ostatecznym raporcie języka z 1983 roku. Wykorzystywane w ADZIE konstrukcje są wynikiem uzyskanych doświadczeń z eksploatacji innych języków programowania. W skrypcie, obok wyczerpującego opisu specyfikacji i implementacji pakietu TEXT-IO, wyjaśniono ideę działania na plikach w tym języku. Wiele przykładów przybliża zrozumienie tych dość trudnych zagadnień. W sumie przestudiowanie tej pracy przynosi uporządkowanie tematyki programowania operacji wejścia-wyjścia, która — jak się okazuje — nie jest tak prosta i trywialna.

Omawiane tutaj książki zostały przygotowane głównie jako opracowania dydaktyczne dla zaawansowanych studentów politechnik kierunku „Informatyka". Prawdziwe będzie jednak stwierdzenie, że nawet bardzo doświadczeni programiści z ośrodków obliczeniowych znajdą w nich wiele interesujących wiadomości, wzbogacających ich wiedzę i umiejętności. Co więcej, w wielu przypadkach treści te umożliwią im opracowanie programów znacznie efektywniejszych w działaniu.

Książki te wnoszą też dodatkową wartość w postaci konsekwentnego wprowadzenia w tę dziedzinę terminologii polskiej. Co prawda nie ze wszystkimi użytymi tu terminami polskimi można się zgodzić, z niektórymi zaś trzeba się oswoić. Interesujący jest więc też dodatek zamieszczony w skrypcie [1], gdzie dokonana jest próba jednolitego przedstawienia terminologii na przykładzie omawianych tam języków. Biorąc pod uwagę chaos terminologiczny, panujący jeszcze w tej dziedzinie, a także często stosowany żargon typu „dumpuję na fajla" — wprowadzenie i stosowanie terminologii polskiej jest konieczne.

Uważam, że należy dążyć do szerszego rozpropagowania tych ważnych w naszym piśmiennictwie pozycji — poprzez ogólnokrajowe wydawnictwa techniczne; po uwzględnieniu uwag czytelników (np. brakuje indeksów ułatwiających odszukiwanie żądanych konstrukcji języka). Na razie zaś pozycje te można zdobyć w księgarni skryptów, znajdującej się w Gmachu Głównym Politechniki Warszawskiej. Trzeba to zrobić szybko, gdyż nakłady są mikroskopijne.

WACŁAW ISZKOWSKI

Spis omawianych skryptów:

[1] Bielecki J.: Programowanie operacji wejścia/wyjścia — ALGOL, FORTRAN, PASCAL. Wydawnictwa Politechniki Warszawskiej, 1983, str. 301, nakład 300 egz.
[2] Bielecki J.: PL/I, programowanie operacji wejścia-wyjścia. Wydawnictwa Politechniki Warszawskiej, 1983, str. 315, nakład 300 egz.
[3] Bielecki J.: System OS/RIAD, opisy prac, zbiory, programy serwisowe. Wydawnictwa Politechniki Warszawskiej, 1984, str. 219, nakład 400 egz.
[4] Bielecki J.: PL/I w praktyce — zbiory indeksowe w systemie OS/360/370, WKiŁ, Warszawa, 1984, (w serii Elektronizacja), str. 42, nakład 4800 egz.
[5] Bielecki J.: VSAM, operacje wejścia-wyjścia w środowisku wirtualnym. Wydawnictwa Politechniki Warszawskiej, 1984 (w druku)
[6] Bielecki J.: ADA — interpretacja operacji wejścia-wyjścia. Wydawnictwa Politechniki Warszawskiej, 1984, (w druku).

Słownik pojęć i terminów z dziedziny grafiki komputerowej (2)

KURSOR (cursor, tracking symbol) — specjalny, poruszający się znak, używany do wskazywania pozycji na ekranie monitora.

LAMPA PAMIĘCIOWA (storage tube) — rodzaj kineskopu, który zachowuje obraz przez długi czas (ok. kilkudziesięciu minut) bez ODNAWIANIA.

LAMPA Z CIEMNYM ŚLADEM (dark trace tube) — rodzaj kineskopu, w którym promieniowanie elektronowe powoduje przyciemnienie zamiast rozjaśnienia powierzchni ekranu. Przykładowo, obraz może być przedstawiony przez oświetlenie z tyłu jako ciemny ślad na jasnym lub półprzezroczystym ekranie lampy.

LAMPA Z ODNAWIANIEM (refresh tube) — rodzaj kineskopu, który wymaga ciągłego wykreślania obrazu (z określoną częstotliwością) w celu utrzymania obrazu stabilnego.

LINIA UKRYTA (hidden line) — odcinek linii przedstawiający niewidoczne krawędzie w dwuwymiarowym rzucie obiektu trójwymiarowego. Linie ukryte mogą być usunięte lub przedstawione jako linie innego rodzaju, koloru lub poziomu jaskrawości.

LUMINOFOR (FOSFOR) (phosphor) — związek chemiczny, który emituje widzialne światło pod wpływem pobudzenia przez promieniowanie elektronowe (p. LAMPA KATODOWA).

MACIERZ ADRESOWALNA (addressable matrix) — macierz utworzona z PUNKTÓW ADRESOWALNYCH określająca WIELKOŚĆ OBRAZU, który można przesłać do GENERATORA OBRAZU.

MACIERZ WIDOKOWA (viewable matrix) — część MACIERZY ADRESOWALNEJ zawierająca PUNKTY ADRESOWALNE rozpinające dyskretną siatkę nad PRZESTRZENIĄ OPERACYJNĄ URZĄDZENIA. Wymiar tej macierzy określa wielkość POWIERZCHNI OBRAZOWANIA wyrażoną w JEDNOSTKACH RASTRA.

MANIPULATOR KULOWY (control ball, track ball, tracker ball) — mechanoelektryczny przekształtnik z kulą, która ma co najmniej dwa stopnie swobody ruchu, służący do PLASOWANIA ELEMENTU OBRAZU na ekranie monitora graficznego. Jest urządzeniem wejściowym monitora.

MANIPULATOR RAMIENIOWY, DRĄŻEK (joystick) — mechanoelektryczny przetwornik z uchwytem, mającym co najmniej dwa stopnie swobody ruchu, służący do PLASOWANIA elementów na ekranie monitora graficznego. Jest urządzeniem wejściowym monitora.

MANIPULATOR TARCZOWY (thumb wheel) — mechanoelektryczny przekształtnik z tarczą lub kołem o jednym stopniu swobody ruchu, służący do PLASOWANIA ELEMENTU OBRAZU na ekranie monitora graficznego wzdłuż jednej ze współrzędnych; często używany w parze z takim samym urządzeniem, sterującym przesunięciami wzdłuż drugiej osi. Jest urządzeniem wejściowym monitora.

MIGOTANIE (blinking) — technika programistyczna lub funkcja sprzętowa urządzenia wyświetlającego, polegająca na przemiennym wyświetlaniu i wygaszaniu ELEMENTU OBRAZU. Zwykle jest używana w celu przyciągnięcia uwagi użytkownika (p. ELEMENT WYRÓŻNIONY).

MIKROGRAFIKA KOMPUTEROWA (computer micrographics) — metody i techniki przekształcania danych graficznych i tekstowych za pomocą komputera na postać lub z postaci mikrofilmu lub mikrofiszy.

MODEL PROBLEMOWY (model) — w dziedzinie grafiki komputerowej opis symboliczny modelu rozpatrywanego problemu, który ma być zobrazowany.

MONITOR (CRT display) — urządzenie wyświetlające, które wykorzystuje LAMPĘ KATODOWĄ.

MONITOR GŁÓWNY (master display, master screen) — dotyczy wielomonitorowych (wieloekranowych) systemów wyświetlania i oznacza ekran, na którym pojawiają się wszystkie wyświetlane obrazy. Zwykle są dołączone do niego różne urządzenia wejściowe, jak klawiatura, manipulator, PIORO itp. (p. MONITOR POMOCNICZY).

MONITOR KONTUROWY (directed beam CRT display device) — GRAFICZNE URZĄDZENIE WYJŚCIOWE z LAMPĄ KATODOWĄ, które wykreśla obraz metodą KONTUROWEGO ROZWINIĘCIA OBRAZU.

MONITOR PLAZMOWY (JARZENIOWY) (plasma panel, gas panel) — GRAFICZNE URZĄDZENIE WYJŚCIOWE składające się z płaskiej płyty wypełnionej gazem, zawierającej siatkę drutów, której węzły po dostarczeniu energii jonizują gaz emitujący światło.

MONITOR POMOCNICZY (slave display, slave screen) — występuje w wielomonitorowym systemie graficznym, z którym jest połączony w taki sposób, że przed każdym przesłaniem informacji należy go programowo zaadresować i włączyć (p. MONITOR GŁÓWNY).

MONITOR REPETYCYJNY (repeater) — monitor sprzężony z MONITOREM GŁÓWNYM w taki sposób, że przedstawia ten sam obraz, lecz nie może być adresowany oddzielnie. Służy do wykonywania kopii lub projekcji na duży ekran.

MONITOR RASTROWY (CRT raster display) — GRAFICZNE URZĄDZENIE WYJŚCIOWE z LAMPĄ KATODOWĄ, które wyświetla obraz metodą RASTROWEGO ROZWINIĘCIA OBRAZU.

MRUGANIE (flicker) — niepożądane MIGOTANIE lub pulsacja obrazu na ekranie monitora pojawiające się wtedy, gdy przy określonym rodzaju LUMINOFORU ,CZĘSTOTLIWOŚĆ ODNAWIANIA jest za mała. MRUGANIE zależy również od JASKRAWOŚCI I KONTRASTU obrazu.

MYSZ MANIPULACYJNA (mouse) — mechanoelektryczny przekształtnik o działaniu MANIPULATORA KULOWEGO lub dwóch MANIPULATORÓW TARCZOWYCH, w obudowie przypominającej mysz, w którym obroty kuli uzyskiwane przez przesuwanie po (dowolnej) powierzchni służą do PLASOWANIA ELEMENTU OBRAZU na ekranie monitora graficznego. Jest urządzeniem wejściowym urządzenia wyświetlającego (p. GRAFICZNE URZĄDZENIE WEJŚCIOWE).

NADĄŻANIE (tracking) — ciąg operacji wykonywanych w monitorze graficznym, powodujących podążanie KURSORA za poruszającym się elementem urządzenia wejściowego (np. PIÓRA ŚWIETLNEGO) lub — określenie pozycji tego elementu na ekranie.

OBCINANIE (clipping, clamping, truncation) — dotyczy pracy urządzenia graficznego (zwykle monitora), gdy współrzędne wykreślanych elementów obrazu przekraczają pojemność rejestrów pozycji X i Y, polega na utrzymaniu maksymalnych zawartości rejestrów, co powoduje zakończenie wykreślania wektorów na brzegu ekranu (p. ZAWIJANIE).

OBIEKT GRAFICZNY (display entity) — uporządkowany zbiór logicznych ELEMENTÓW OBRAZU, którymi można operować jako całością.

OBRAZ CYFROWY (RASTROWY) (raster image) — obraz złożony z KOMÓREK OBRAZU, dla których zdefiniowana jest funkcja dyskretna (np. jaskrawość, szarość) przynajmniej o dwóch stanach (poziomach jaskrawości, szarości).

OBRAZ EKSPONOWANY (image) — zbiór ELEMENTÓW OBRAZU w PRZESTRZENI OBRAZOWANIA.

OBRAZ FIZYCZNY (display) — wykreślony na POWIERZCHNI OBRAZOWANIA KADR OBRAZU (wizualne przedstawienie danych).

OBRAZ KONTUROWY (RYSUNEK LINIOWY) (coded image) — obraz złożony z linii i punktów, których dane mają ściśle określoną wewnętrzną strukturę, na ogół sekwencyjną (patrz: DANE GRAFICZNE, def. 2).

OBRAZOWANIE (displaing) — wizualne przedstawianie danych.

OBRAZOWANIE GŁĘBI (depth quening) — przedstawianie głębi obrazu trójwymiarowego, polegające zwykle na modulowaniu jaskrawości linii zgodnie z jej sugerowaną odległością od obserwatora. Może odnosić się również do perspektywy, stereoskopii i obrazowania efektów ruchu.

ODŚWIEŻANIE, ODNAWIANIE (regeneration, refresh) — proces powtarzalnego wykreślania KADRU OBRAZU na ekranie monitora. Czas świecenia LUMINOFORU LAMPY KATODOWEJ pobudzonego jednokrotnym wykreśleniem obrazu jest krótki (ułamki sekundy), dlatego utrzymanie stabilnego obrazu wymaga jego ciągłego wykreślania z częstotliwością ok. 50 Hz (p. CZĘSTOTLIWOŚĆ ODNAWIANIA, MRUGANIE).

OKIENKOWANIE (windowing) — wyizolowanie dowolnej części wyświetlonego obrazu, przesunięcie jej na środek ekranu i wyskalowanie w takiej proporcji, aby zajęła cały ekran. W technice filmowej tę czynność nazywa się kadrowaniem.

OKNO (window, logical screan) — ograniczony obszar w obrębie PRZESTRZENI OBRAZOWANIA, który jest odwzorowywany na WZIERNIK. Może być rozszerzony do całej PRZESTRZENI OBRAZOWANIA.

OKRAWANIE (scissoring) — w monitorze, który dopuszcza rysowanie obrazów większych od rozmiarów ekranu, dotyczy reakcji na przekroczenie obszaru ekranu przez strumień elektronowy i oznacza automatyczne wygaszenie strumienia, gdy wychodzi poza ramy ekranu. Przy braku wygaszenia strumienia występują odbicia obrazu.

PANEL GRAFICZNY (display panel) — KONSOLA GRAFICZNA oprogramowana w sposób umożliwiający pracę w TRYBIE INTERAKCYJNYM.

PANORAMA DYNAMICZNA (panning) — stopniowe przemieszczanie OKNA w PRZESTRZENI OBRAZOWANIA, tak

aby we WZIERNIKU uzyskać wrażenie bocznego ruchu obrazu (szczególny rodzaj PRZEGLĄDANIA).

PIÓRO (stylus) — 1) wskaźnik używany do PLASOWANIA ELEMENTU OBRAZU, np. PIÓRO ŚWIETLNE, PIÓRO ULTRADŹWIĘKOWE itp., 2) rylec, element GŁOWICY KREŚLAKA służący do nanoszenia konturów na twardym podłożu.

PIÓRO NAPIĘCIOWE (voltage gradient stylus) — PIÓRO, którego pozycja jest określona przez poziom napięcia mierzonego na oporowej siatce. Jest urządzeniem wejściowym KONSOLI GRAFICZNEJ, wykorzystywanym zazwyczaj w połączeniu z RYSOWNICĄ.

PIÓRO ŚWIETLNE (light pen, selector pen) — czujnik fotoelektryczny wykonany w kształcie PIÓRA, który generuje impuls elektryczny gdy w jego polu widzenia znajduje się świecący element. Zazwyczaj służy do WYKRYWANIA ELEMENTU OBRAZU na ekranie monitora.

PIÓRO ULTRADŹWIĘKOWE (sonic pen) — wejściowe urządzenie KONSOLI GRAFICZNEJ wykonane w postaci PIÓRA, którego pozycja (np. na ekranie monitora, RYSOWNICY) jest określana za pomcą sygnałów ultradźwiękowych.

PLAMKA KIERUNKOWA (aiming symbol, aiming circle, aiming field) — plamka świetlna rzucana przez PIÓRO ŚWIETLNE na powierzchnię ekranu monitora, pomagająca w dokładnym ustawieniu PIÓRA i umiejscowieniu jego pola widzenia.

PLASOWANIE (plate) — wprowadzanie współrzędnych położenia ELEMENTU OBRAZU na POWIERZCHNI OBRAZOWANIA za pomocą GRAFICZNEGO URZĄDZENIA WEJŚCIOWEGO, np. MANIPULATORA KULOWEGO (przeciwieństwo WYKRYWANIA).

PŁASZCZYZNA RZUTOWANIA (viewing plane) — dwuwymiarowa powierzchnia, na którą jest rzutowana PRZESTRZEŃ MODELU. Może być rozszerzona do trójwymiarowej przestrzeni rzutowania.

POCZĄTEK (origin) — dowolny punkt odniesienia wybrany przez użytkownika w układzie współrzędnych mający wszystkie współrzędne równe zeru.

PODPOWIADANIE (prompting) — informowanie użytkownika o czynnościach możliwych do wykonania w następnym kroku; W GRAFICE KOMPUTEROWEJ polega na wyświetlaniu komunikatów lub WYKAZU FUNKCJI na ekranie monitora, a także na zapaleniu odpowiednich lampek na KLAWIATURZE FUNKCYJNEJ monitora.

PODŚWIETLANIE SZABLONU (forms flash, flashing) — okresowe wyświetlanie SZABLONU jako TŁA OBRAZU.

POWIERZCHNIA OBRAZOWANIA (display surface) — nośnik w GRAFICZNYM URZĄDZENIU WYJŚCIOWYM na którym ELEMENT KREŚLĄCY OBRAZ tworzy OBRAZ FIZYCZNY (np. papier, film, ekran lampy katodowej).

POWTARZALNOŚĆ (repeatability) — miara sprzętowej dokładności wykreślania obrazu, w tym samym miejscu ekranu monitora, w kolejnych cyklach ODNAWIANIA.

POZIOM INTENSYWNOŚCI ŚWIECENIA (intensity level) — ustawiany programowo dyskretny poziom natężenia światła emitowanego przez LAMPĘ KATODOWĄ MONITORA GRAFICZNEGO.

POZOSTAWIANIE ŚLADU (inking) — generowanie linii ciągłej wzdłuż drogi wyznaczonej przez GRAFICZNE URZĄDZENIE WEJŚCIOWE (np. PIÓRO ŚWIETLNE).

POZYCJA BIEŻĄCA (current position) — położenie ELEMENTU KREŚLĄCEGO OBRAZ wynikające z ostatnio wykonanego ROZKAZU GRAFICZNEGO. Może być wyrażone we współrzędnych PRZESTRZENI MODELU lub PRZESTRZENI OBRAZOWANIA.

POZYCJONOWANIE (positioning) — ustawienie ELEMENTU KREŚLĄCEGO OBRAZ w ustalonym miejscu POWIERZCHNI OBRAZOWANIA, np.: promienia elektronowego w LAMPIE KATODOWEJ lub GŁOWICY KREŚLAKA na stole lub bębnie.

PROGRAMOWA KLAWIATURA FUNKCYJNA (program function keyboard) — 1) zbiór PROGRAMOWYCH KLUCZY FUNKCYJNYCH, 2) klawiatura, której klawisze nie mają na stałe przypisanych funkcji, a ich ustalenie jest zależne od wykonywanego programu.

→

Kierzkowski Z., Maluszyński J.: Zarządzanie współbieżną aktualizacją bazy danych (1)

INFORMATYKA 1985, nr 3, s. 1

Pierwsza część charakterystyki rozwiązań zapewniających zarządzanie współbieżną aktualizacją bazy danych. Omówiono wybrane aspekty funkcjonowania oprogramowania zarządzającego taką aktualizacją.

Iszkowski W.: Przegląd mechanizmów komunikacji

INFORMATYKA 1985, nr 3, s. 3

Charakterystyka podstawowych mechanizmów komunikacji służących do synchronizacji procesów współbieżnych. Omówiono cechy i rozwiązania występujące w najczęściej spotykanych wersjach tych mechanizmów.

Litwiniuk A. I.: Kompilator LOGLANU 82 dla MERY 400

INFORMATYKA 1985, nr 3, s. 7

Szczegółowe omówienie konstrukcji, sposobu działania i parametrów eksploatacyjnych kompilatora języka programowania LOGLAN 82 dla minikomputera MERA 400.

Jabłoński K.: Język programowania BCPL (2)

INFORMATYKA 1985, nr 3, s. 11

Dokończenie charakterystyki języka BCPL, zawierające omówienie przykładowego programu oraz zasad implementacji.

Кежковски З., Малушиньски Я.: Управление параллельным обновлением базы данных (1)

INFORMATYKA 1985, № 3, стр. 1

Первая часть характеристики решений обеспечивающих управление параллельным обновлением базы данных. Обсуждены избранные стороны функционирования программного обеспечения управляющего таким обновлением.

Ишковски В.: Обзор механизмов связи

INFORMATYKA 1985, № 3, стр. 3

Характеристика основных механизмов связи служащих для синхронизации параллельных процессов. Обсуждены свойства и решения выступающие в чаще всего встречаемых вариантах этих механизмов.

Литвинюк А. И.: Компилятор языка LOGLAN 82 для мини-ЭВМ MERA 400

INFORMATYKA 1985, № 3, стр. 7

Подробное обсуждение конструкции, способа действия и эксплуатационных параметров компилятора языка программирования LOGLAN 82 для мини-ЭВМ MERA 400.

Яблоньски К.: Язык программирования BCPL (2)

INFORMATYKA 1985, № 3, стр. 11

Завершение характеристики языка BCPL, содержащее обсуждение примера программы и правил реализации.

Kierzkowski Z., Maluszyński J.: Management of concurrent data base updating (1)

INFORMATYKA 1985, No. 3, p. 1

First part of characteristics of solutions, which secure management of concurrent data base updating. Selected features of software for management of such updating are discussed.

Iszkowski W.: Survey of communication mechanisms

INFORMATYKA 1985, No. 3, p. 3

Characteristics of basic communication mechanisms which are applied for concurrent processes synchronization. Features and solutions appearing in main versions of those mechanisms are discussed.

Litwiniuk A. I.: LOGLAN 82 compiler for MERA 400

INFORMATYKA 1985, No. 3, p. 7

Detailed discussion of construction, operation and operating parameters of the LOGLAN 82 compiler for MERA 400 minicomputer.

Jabłoński K.: BCPL programming language (2)

INFORMATYKA 1985, No. 3, p. 11

Conclusion of the BCPL language characteristics, which include discussion of examplary program and implementation rules.

Kierzkowski Z., Maluszyński J.: Verwaltung von simultaner Aktualisierung einer Datenbank (1)

INFORMATYKA 1985, Nr. 3, S. 1

Erster Teil einer Charakteristik von Lösungen, die die Verwaltung simultaner Aktualisierung einer Datenbank sichern. Es wurden ausgewählte Aspekte der Funktionierung von Software für Verwaltung einer solchen Aktualisierung besprochen.

Iszkowski W.: Übersicht von Kommunikationsmechanismen

INFORMATYKA 1985, Nr. 3, S. 3

Eine Charakteristik der grundlegenden Kommunikationsmechanismen, die zur Synchronisierung der simeltanen Prozesse dienen. Es wurden Eigenschaften und Lösungen in den am häufigsten begegneten Versionen dieser Mechanismen besprochen.

Litwiniuk A. I.: LOGLAN 82 Kompilierer für MERA 400

INFORMATYKA 1985, Nr. 3, S. 7

Detaillierte Besprechung der Konstruktion, Wirkungsweise und Betriebsparameter von LOGLAN 82 Kompilierer für MERA 400 Kleinrechner.

Jabłoński K.: BCPL — Programmiersprache (2)

INFORMATYKA 1985, Nr. 2, S. 11

Beendigung einer Charakteristik der BCPL-Sprache, die Besprechung eines Beispielprogrammes und Regeln für Implementierung umfasst.

Terminologia

PROGRAMOWY KLAWISZ FUNKCYJNY (program function key) — klawisz wchodzący w skład wykazu FUNKCYJNEGO, wybierany zwykle za pomocą PIÓRA ŚWIETLNEGO, wywołujący ciąg instrukcji programu, powodujących identyczny skutek jak sygnał inicjowany przez użycie KLAWISZA FUNKCYJNEGO.

PRZEDNI PLAN OBRAZU (display foreground, foreground image, dynamic image) — zbiór tych ELEMENTÓW OBRAZU FIZYCZNEGO, które są zmieniane przez program lub użytkownika.

PRZEGLĄDANIE (scrolling) — przemieszczanie OKNA w PRZESTRZENI OBRAZOWANIA w taki sposób, że we WZIERNIKU pojawiają się niewidoczne dotychczas elementy OBRAZU EKSPONOWANEGO.

PRZESŁANIANIE (shielding, reverse clipping) — usuwanie wszystkich ELEMENTÓW OBRAZU lub ich części, które leżą wewnątrz określonego obszaru POWIERZCHNI OBRAZOWANIA.

PRZESTRZEŃ MODELU (model space) — przestrzeń zdefiniowana przez układ współrzędnych wprowadzony do opisu MODELU PROBLEMOWEGO.

PRZESTRZEŃ OBRAZOWANIA (image space, virtual space) — obszar zdefiniowany przez układ współrzędnych na PŁASZCZYŹNIE RZUTOWANIA, który jest odwzorowywany w PRZESTRZEŃ URZĄDZENIA GRAFICZNEGO.

PRZESTRZEŃ OPERACYJNA URZĄDZENIA (display space, operating space) — obszar w PRZESTRZENI URZĄDZENIA GRAFICZNEGO, którego zawartość jest przedstawiona na POWIERZCHNI OBRAZOWANIA.

PRZESTRZEŃ URZĄDZENIA GRAFICZNEGO (device space) — obszar zdefiniowany przez układ współrzędnych na POWIERZCHNI OBRAZOWANIA, ograniczony pojemnością rejestrów pozycji X, Y w urządzeniu graficznym.

PRZESUW (move) — w grafice przemieszczenie ELEMENTU KREŚLĄCEGO OBRAZ z jednego położenia w drugie.

PRZEWIJANIE (rolling) — technika manipulacji (przeglądania) danych tekstowych, polegająca na usuwaniu (ciągłym lub skokowym) górnego wiersza wyświetlanych danych, przesunięciu pozostałych wierszy w górę i wstawieniu nowego wiersza na dole ekranu. Przesunięcie może odbywać się także w dół.

PUNKT ADRESOWALNY (addressable point) — dowolna pozycja PRZESTRZENI URZĄDZENIA GRAFICZNEGO, do której może zostać skierowany ELEMENT KREŚLĄCY OBRAZ (pióro KREŚLAKA lub strumień elektronowy monitora), określona przez WSPÓŁRZĘDNE. Istnieje skończona liczba pozycji adresowalnych które mają postać dyskretnej siatki w PRZESTRZENI URZĄDZENIA GRAFICZNEGO.

WOJCIECH MOKRZYCKI
LENA WALKIEWICZ

■ Firma BRITISH AEROSPACE oraz THORN EMI zakończyły ostatnio program badawczy, który — zdaniem zainteresowanych — może spowodować znaczne zwiększenie możliwości stosowania robotów w przemyśle lotniczym. Realizacja badań doprowadziła mianowicie do zatrudnienia pierwszego, ciężkiego antropomorficznego robota w zakładach BRITISH AEROSPACE w Filton (Wielka Brytania). Robot ten nazwany WORKMASTER zastępuje ludzi w niebezpiecznej pracy, jaką jest czyszczenie wnętrza ogona samolotu bojowego F-111 przy użyciu, wyrzucanego z dużym ciśnieniem, strumienia wody (wnętrze ogona wykorzystywane jest jako jeden ze zbiorników paliwa). Dalsze prace dotyczą możliwości stosowania robota do profilowania — również za pomocą sprężonego strumienia wody — materiałów syntetycznych służących jako wykładziny kabin samolotów. Robot zainstalowany jest na specjalnym stanowisku roboczym, a pracę wykonuje bez jakiejkolwiek pomocy człowieka. (W)

•

■ Minikomputery wciąż tanieją. Firma WHITECHAPEL COMPUTER WORKS (WCW) zaoferowała ostatnio superminikomputer o szokująco niskiej cenie. Młoda, bo istniejąca od roku, firma WCW proponuje minikomputer MG-1 za 5495 funtów, podczas gdy ogólnie przyjęte ceny sprzętu podobnej klasy są rzędu 20 tys. funtów. MG-1 zawiera najnowszy procesor firmy NATIONAL SEMICONDUCTOR — 32016. Wybór tego procesora nie był przypadkowy, wchodzi on bowiem w skład jedynego obecnie na rynku elektronicznym zestawu obejmującego procesor, blok zarządzania pamięcią wirtualną oraz współprocesor operacji zmiennoprzecinkowych. Podstawowa wersja superminikomputera, nazwana MG-1/10, ma 512 KB pamięci RAM, dysk Winchester o pojemności 10 MB, dysk elastyczny 0,8 MB oraz monitor o rozdzielczości ekranu 1024××800. MG-1/10 wyposażony jest w system operacyjny UNIX; ma też tzw. myszkę, ułatwiającą wyszukiwanie informacji. Drożej, bo za 7495 funtów oferowana jest wersja MG-1/45, wyposażona w dysk Winchester o pojemności 45 MB. (W)

•

■ Walka o palmę pierwszeństwa dotyczy już komputerów piątej generacji. Rząd federalny Australii przeznaczył w roku finansowym 1984—1985, 250 tys. dolarów austral. na rozpoczęcie prac badawczych nad Piątą Generacją komputerów. Większość specjalistów uważa jednak tą sumę za małą, a cały projekt za spóźniony. Dla porównania, japońska firma MITI ma zamiar wykorzystać na ten sam cel 450 mln dol. austral., a rząd Wielkiej Brytanii przeznaczył 350 mln dol. austral. Sumy te mają pokryć koszty badań planowanych na dziesięć najbliższych lat. W USA wyodrębniono natomiast dwa zespoły zajmujące się Piątą Generacją. Jeden zespół, będący konsorcjum dwunastu firm amerykańskich, ma zamiar sumą 75 mln dol. ameryk. sfinansować dziesięć lat badań. Pracom drugiego zespołu patronuje Pentagon, który wyasygnował na ten cel sumę 1 mld dol. austral. (W)

•

■ Grupa naukowców z British Columbia University w Kanadzie stworzyła pierwszy system oprogramowania umożliwiający szybkie i precyzyjne programowanie robotów przemysłowych z wykorzystaniem grafiki przestrzennej. System przystosowany jest do programowania robotów spawających. Po utworzeniu na ekranie monitora trójwymiarowego obrazu przedmiotu spawanego, operator wskazuje linie spawów. Instrukcje operatora są weryfikowane i w przypadku pomyłki, takiej jak np. rozkaz dwukrotnego spawu, generowany jest sygnał dźwiękowy. Program analizuje kolejne operacje, identyfikuje ruchy poszczególnych fragmentów łańcucha kinematycznego oraz określa warunki spawania. System został zaimplementowany na komputerze DEC VAX 11/750. (W)

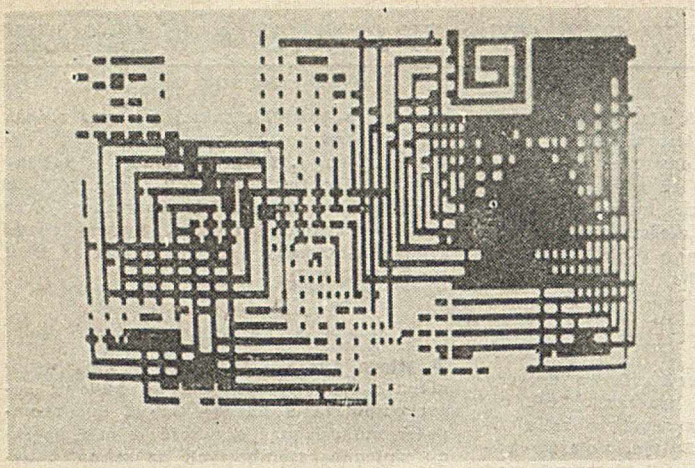
•

■ Młoda firma ETA Systems Inc., wyłoniona z CDC, przystąpiła do budowy superkomputera o nazwie GF-10. Superkomputer ma osiągnąć szybkość 10 mld operacji zmiennoprzecinkowych na sekundę. Jeszcze niedawno sensacje budziło oświadczenie firmy HITACHI o pracach nad komputerem o szybkości 630 mln operacji. Olbrzymia szybkość maszyn GF-10 ma zostać zrealizowana za pomocą ośmiu pracujących równolegle procesorów. Wprawdzie rzeczywiste problemy obliczeniowe zmniejszą szybkość do 2—3 mld operacji zmiennoprzecinkowych na sekundę, ale mimo wszystko jest to 20—30 razy więcej w stosunku do możliwości maszyn CYBER-205 czy CRAY-1. Natomiast Seymour Cray, założyciel firmy CRAY, zakomunikował, że budowany przez firmę superkomputer będzie pracował 48 razy szybciej niż CRAY-1, a więc z szybkością 4,8 mld operacji zmiennoprzecinkowych na sekundę. Mimo że nie sprecyzowano, czy chodzi tu o szybkości maksymalne czy też o średnie, informacja ta wzbudziła sensację. (W)

•

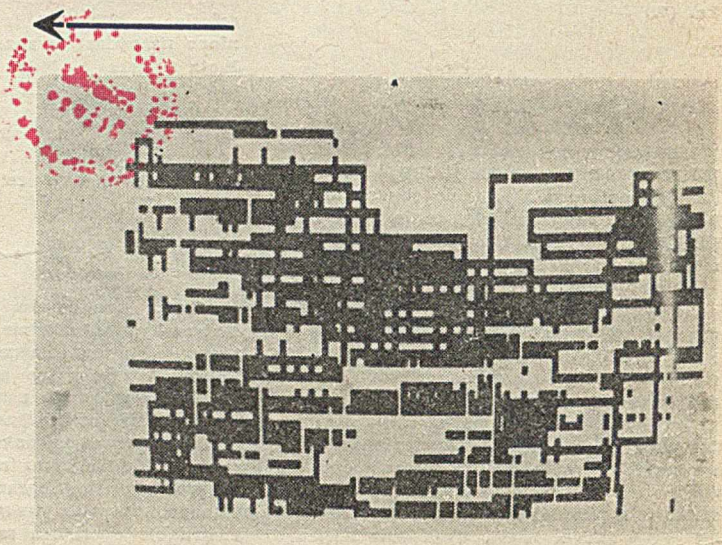
■ W Jugosławii znajduje się obecnie ok. 120 tys. komputerów osobistych. Władze oświatowe podjęły decyzję o wprowadzeniu mikrokomputerów do szkół. W Serbii eksperymentalne zajęcia zaczynają się w 12 szkołach i obejmują uczniów klas ósmych. W Słowenii we wszystkich szkołach ponadpodstawowych prowadzi się zajęcia z informatyki. (N)

W SKRÓCIE • W SKRÓCIE • W SKRÓCIE • W SKRÓCIE • W SKRÓCIE • W SKRÓCIE



```
5 REM PROGRAM "SPIRALA"
10 DEFINT A-Z
20 'ANBOM
30 CLS                                  :REM ZEROWANIE EKRANU
40 V=1
50 X=RND(127)                           :REM GENERACJA
60 Y=RND(47)                            :REM PUNKTU STARTOWEGO
70 V=-V
80 L=0
90 L=L+1
100 DX=0                                :REM W GORE
110 DY=-1
120 D=L
130 GOSUB 300
140 L=L+1
150 DX=1                                :REM W PRAWO
160 DY=0
170 D=L+L
180 GOSUB 300
190 L=L+1
200 DX=0                                :REM W DOL
210 DY=1
220 D=L
230 GOSUB 300
240 L=L+1
250 DX=-1                               :REM W LEWO
260 DY=0
270 D=L+L
280 GOSUB 300
290 GOTO 90
300 FOR J=1 TO D                        :REM RYSOWANIE LINII
310 X=X+DX
320 Y=Y+DY
330 IF X=0 OR Y=0 OR X=128 OR Y=48 GOTO 50   :REM BRZEG EKRANU
340 IF V=1 THEN SET(X,Y)                :REM ZAPALENIE PUNKTU
350 IF V=-1 THEN RESET(X,Y)             :REM GASZENIE PUNKTU
360 NEXT J
370 RETURN
380 END
```

```
5 REM PROGRAM "LABIRYNT"
10 DEFINT A-Z
20 RANDOM
30 CLS                                  :REM ZEROWANIE EKRANU
40 V=1
50 X=RND(127)                           :REM GENERACJA
60 Y=RND(47)                            :REM PUNKTU STARTOWEGO
70 V=-V
80 DX=0                                 :REM W GORE
90 DY=-1
100 K=1
110 GOSUB 250
120 DX=1                                :REM W PRAWO
130 DY=0
140 K=3
150 GOSUB 250
160 DX=0                                :REM W DOL
170 DY=1
180 K=1
190 GOSUB 250
200 DX=-1                               :REM W LEWO
210 DY=0
220 K=3
230 GOSUB 250
240 GOTO 80
250 FOR J=1 TO K*RND(12)                :REM RYSOWANIE LINII
260 X=X+DX
270 Y=Y+DY
280 IF X=0 OR Y=0 OR X=128 OR Y=48 GOTO 50   :REM BRZEG EKRANU
290 IF V=1 THEN SET(X,Y)                :REM ZAPALENIE PUNKTU
300 IF V=-1 THEN RESET(X,Y)             :REM GASZENIE PUNKTU
310 NEXT J
320 RETURN
330 END
```



Spirala i labirynt

Przedstawione grafiki (zdjęcia ekranu monitora) otrzymane są przy użyciu mikrokomputera MERITUM.

Pierwsza z nich powstała poprzez naprzemienne wrysowywanie lub wymazywanie spiralnych łamanych o losowo generowanych punktach początkowych.

Druga grafika powstała poprzez wrysowywanie (lub wymazywanie) odcinków o losowej długości kolejno w górę, w prawo, w dół, w lewo, etc. — aż do osiągnięcia krawędzi ekranu, po czym następuje generacja kolejnej łamanej o losowo wybranym punkcie początkowym.

Oba programy wykorzystują bardzo podobny podprogram — umożliwiający wykreślenie odcinka o określonej lub losowej długości w jednym z ośmiu kierunków „róży wiatrów", w zależności od parametrów DX i DY.

W wydrukach obu programów (BASIC dla MERITUM I) instrukcje SET X, Y i RESET X, Y odpowiednio zapalają lub gaszą na ekranie punkt o współrzędnych X i Y. Cały ekran jest matrycą liczącą 47 wierszy po 127 elementów.

ZBIGNIEW SZKARADNIK