mikroKLAN 22

10 1985

informatyka

Egzekutor E6RM dla R-32

RATFOR

Złożoność algorytmów

WYDAWNICTWA NAUKOWO-TECHNICZNE

Cena 100 zł

informatyka

10 1985

Nr 10
Miesięcznik Rok XX
Październik 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. 0911-1300/85. Obj. 4,0 ark. druk. Nakład 6400 egz. P-78.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 100 zł
Prenumerata roczna 1200 zł

W NUMERZE:

W NASTĘPNYCH NUMERACH:

- Stanisław Gasik o wprowadzeniu do programometrii
- Zbigniew Szkaradnik porównuje języki programowania
- Wiesław Nosowski i Bolesław Szomański o pakiecie symulacyjnym
- Jan Bielecki o programowaniu asemblerowym w języku fig-FORTH
- Jerzy Sukiennik o rozrachunku gospodarczym zakładowego ośrodka informatyki
- Edward Bieleninik i inni o podsieci komunikacyjnej Międzyuczelnianej Sieci Komputerowej
- Ian Pyle o pakiecie do specyfikacji programów w Adzie
- Marek Sikora o modułowym systemie mikroprocesorowym kompatybilnym z systemem MIKROSTER
- Jerzy Zakręcki o abstrakcjach w programowaniu

KRZYSZTOF PERYCZ
ZETO — Gdańsk

Egzekutor E6RM dla R-32

Symulator i emulator ODRY 1305 na komputerze R-32 oraz ich geneza zostały opisane w publikacjach [2] i [3]. Narzędzia te umożliwiają posiadaczom instalacji R-32 bezpośrednie wykorzystanie istniejących programów czy też całych systemów informatycznych, eksploatowanych dotąd na ODRZE z zastosowaniem egzekutora E6RM oraz systemu operacyjnego GEORGE-2.

Symulator i emulator są pod względem funkcjonalnym całkowicie zgodne, a różnią się jedynie sposobem instalowania oraz szybkością. Emulator jest 4-krotnie szybszy od symulatora, lecz dla zainstalowania wymaga niewielkiej modyfikacji pamięci stałej R-32.

To, że symulator, a potem emulator, zostały zrealizowane tak małym nakładem sił i środków [1]) oraz wykazują dużą niezawodność działania, wynika bezpośrednio z przyjętych założeń.

ROZWAŻANE WARIANTY REALIZACJI

Pierwszą decyzją, która pociągnęła za sobą niemal wszystkie następne, było ustalenie zakresu symulacji procesora ODRY 1305. Symulowanie całego procesora ODRY, łącznie z instrukcjami uprzywilejowanymi [6], umożliwiałoby co prawda bezpośrednie przenoszenie dowolnych egzekutorów, ale wtedy realizacja ich funkcji krytycznych z punktu widzenia czasu podlegałaby znacznemu wydłużeniu, wskutek nieuniknionych strat spowodowanych przez symulację.

Wybranie wariantu, w którym następowałaby symulacja tylko nieuprzywilejowanych instrukcji ODRY („NORMAL MODE"), implikuje konieczność dodatkowego zrealizowania funkcji wybranego egzekutora w kodzie komputera R-32. Zaletą takiego rozwiązania jest możliwość uzyskania dużej prędkości działania egzekutora, natomiast jego wadami są „przypisanie" do konkretnego egzekutora oraz konieczność napisania w kodzie R-32 całego egzekutora bądź jego części. Oczywiście, pozostawało jeszcze do zadecydowania, czy wybrać egzekutor E6RM czy też EWGN, umożliwiający eksploatację systemu operacyjnego GEORGE-3. Ponieważ GEORGE-3 odsłania w pełni swoje zalety dopiero wtedy, gdy działa na szybkich maszynach z możliwościami zdalnego dostępu, czego nie mogłyby zapewnić instalacje R-32 obciążone symulatorem, wybór padł na egzekutor E6RM. Za egzekutorem E6RM przemawiała dodatkowo utrzymująca się wciąż jego popularność. Obawy budziła tylko jego obszerność. Jest to, jak wiadomo, duży nakładkowany egzekutor, realizujący wiele skomplikowanych funkcji [7]. Ewentualne błędy w realizacji tych funkcji na R-32 mogłyby skutecznie uniemożliwić uzyskanie kompatybilności z ODRĄ, działającą pod oryginalnym egzekutorem E6RM, zaś wysiłek w wyszukaniu i skorygowaniu tych błędów byłby tak znaczny, że całe przedsięwzięcie stałoby się nieopłacalne.

W tej fazie tworzenia założeń, najistotniejszym i zarazem najszczęśliwszym okazał się pomysł wypływający z zaobserwowania następujących faktów:

- część stała egzekutora E6RM zawierająca wszystkie funkcje krytyczne z punktu widzenia czasu jest stosunkowo niewielka i tylko ona wykonuje się w stanie „EXEC MODE" (w stanie tym korzysta się z instrukcji uprzywilejowanych)
- część nakładkowana wykonuje się w stanie „NORMAL MODE" procesora i zawiera zdecydowaną większość funkcji egzekutora [4]

Wspomniany pomysł polegał na zrealizowaniu w kodzie R-32 tylko stałej części egzekutora E6RM, z zachowaniem styku programowego z częścią nakładkowaną, tak by można było bezpośrednio korzystać z oryginalnych nakładek E6RM (w kodzie ODRY). Pozwalało to osiągnąć potrójny efekt:

- olbrzymią oszczędność pracochłonności, wynikającej z poniechania przekodowywania nakładek na kod R-32
- dużą niezawodność działania
- pełną kompatybilność, wynikającą z użycia sprawdzonych, oryginalnych nakładek.

Z rozważań teoretycznych wynikało, że zwolnienie działania komputera, wynikłe z symulacyjnego wykonywania nakładek, nie będzie miało odczuwalnego wpływu na pogorszenie się parametrów eksploatacyjnych egzekutora (co później sprawdziło się w praktyce).

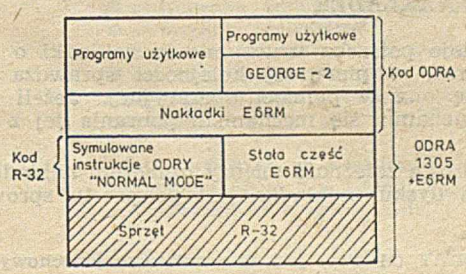
PRZYJĘTE ZAŁOŻENIA

Powyższe obserwacje w decydujący sposób przyczyniły się do przyjęcia dla symulatora E6RM-S następujących założeń:

- symulacja tylko instrukcji „NORMAL MODE" ODRY 1305
- realizacja w kodzie R-32 części stałej egzekutora E6RM, z utrzymaniem styku programowego z jego częścią nakładkowaną, zrealizowaną w kodzie ODRA.

Symulator został ponadto zaprojektowany jako program niezależny, tj. działający na R-32 bez jakiegokolwiek systemu operacyjnego. Motywacja takiego rozwiązania była trojaka: po pierwsze — technika symulacyjna na tyle obciąża arytmometr, że równoległe przetwarzanie zadań „riadowskich" nie byłoby opłacalne; po drugie — powszechnie spotykane konfiguracje R-32, ze względu na swoją szczupłość, w praktyce uniemożliwiają równoległą eksploatację zadań „odrowskich" i „riadowskich"; po trzecie wreszcie — większa część niezbędnych funkcji systemu operacyjnego (w postaci gotowych nakładek) została już zrealizowana, a więc mniej kłopotliwe było ich uzupełnienie o brakującą część stałą niż wkomponowanie całości w system operacyjny OS lub DOS.

Przyjęte założenia w sposób schematyczny obrazuje rys. 1. Przyjęta kolejność realizacji (najpierw symulator, potem emulator) umożliwiła z kolei wyselekcjonowanie tych krytycznych z punktu widzenia czasu części symulatora, które najbardziej opłaca się przyspieszyć drogą wspomagania sprzętowego. Po przeprowadzeniu analizy, wybór padł na pętlę pobrania i deszyfracji rozkazu ODRA. Rozważenie możliwości sprzętowych R-32 [8] wykazało, że najlepszy efekt przyniesie umieszczenie tej pętli w postaci ciągu specjalnych mikrorozkazów w pamięci stałej R-32, bez konieczności tworzenia dodatkowych „przystawek".



Rys. 1. Struktura symulatora E6RM-S

[1]) Prace nad symulatorem E6RM-S rozpoczęły się w kwietniu 1982 r. i trwały do listopada 1982 r., natomiast nad emulatorem E6RM-E — w maju 1983 r. i trwały do października 1983 r. W obu przypadkach całość prac prowadził zespół autorski z ZETO Gdańsk w następującym składzie: mgr inż. Marek Dziedzic, mgr inż. Krzysztof Perycz i mgr inż. Jerzy Wiliński. Emulator E6RM-E jest zgłoszony w Urzędzie Patentowym PRL pod numerem P248467.

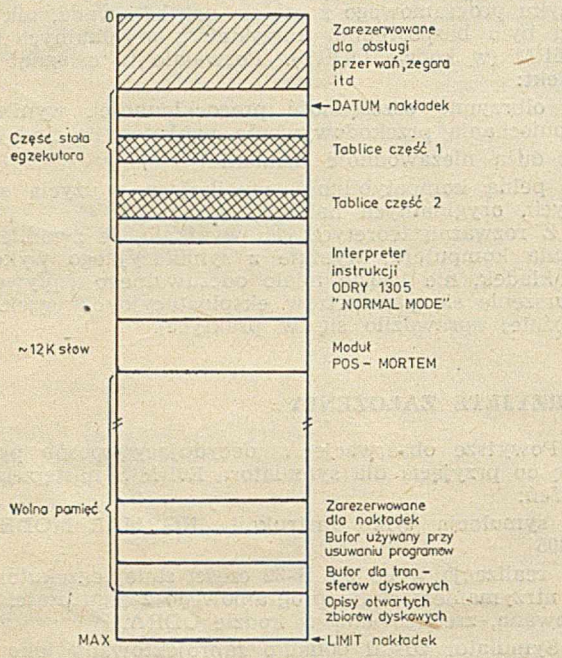
Jest to drugi bardzo istotny moment, gdyż inną drogą zakończenie całości przedsięwzięcia w tak wąskim zespole autorskim byłoby nierealne.

Używane w dalszej części artykułu określenie „emulator" będzie dotyczyło zarówno emulatora E6RM-E, jak i symulatora E6RM-S.

ORGANIZACJA PAMIĘCI OPERACYJNEJ EMULATORA

Na rys. 2 przedstawiono obraz pamięci operacyjnej emulatora. Pamięć o niższych adresach jest zajęta przez część stałą egzekutora, interpreter instrukcji „NORMAL MODE" ODRY oraz moduł POST-MORTEM, obsługujący sytuacje wyjątkowe.

Styk programowy z oryginalną częścią nakładkowaną stanowią tablice (zawierające m. in. aktualny stan urządzeń we-wy oraz zbiorów) umieszczone w części stałej egzekutora, pod ściśle określonymi adresami, w formacie ODRY [2]).



Rys. 2. Obraz pamięci emulatora

Tablice te są dostępne dla części nakładkowanej, ze względu na to, że DATUM dla nakładek jest odpowiednio ustawione.

Programy użytkowe ładowane są do wolnego obszaru pamięci — od niskich adresów wzwyż. Każdy program ma swój 128-słowowy „blok informacyjny", usytuowany bezpośrednio pod DATUM programu. W pamięci może przebywać jednocześnie wiele programów. Gdy jeden z nich jest usuwany lub zmienia swój rozmiar, pozostałe programy są odpowiednio przesuwane. Tą drogą unika się fragmentacji pamięci operacyjnej.

Górna część pamięci, nie zajęta przez programy, jest powiązana w łańcuch bloków po 256 słów, zaczynający się bezpośrednio pod tablicą otwartych zbiorów, liczącą 512 słów, a kończący się bezpośrednio nad LIMIT ostatniego załadowanego programu użytkowego. Ta część pamięci używana jest do przechowywania nakładek egzekutora (pierwsze trzy bloki od góry mają dodatkowe przeznaczenie i nie są nigdy używane do przechowywania programów — patrz rys. 2).

OBSŁUGA NAKŁADEK

Gdy istnieje potrzeba uruchomienia nakładki o określonym numerze, w pierwszej kolejności sprawdza się czy znajduje się ona w pamięci operacyjnej. Jeżeli jej tam nie ma, uruchamia się mechanizm pobrania jej z pamięci dyskowej.

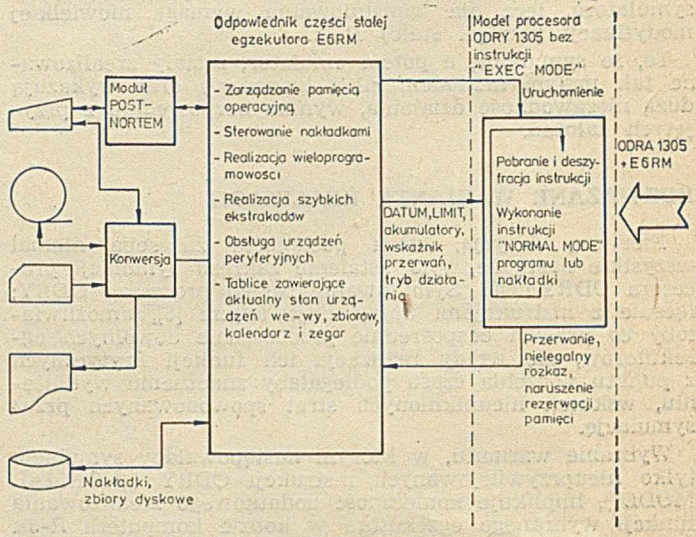
Istnieje prosta zależność między numerem nakładki, a jej adresem na dysku: nakładka jest po prostu sprowadzana w dogodne miejsce w wolnym obszarze pamięci. Może to wprawdzie spowodować usunięcie innej nakładki, lecz w miarę możliwości wyszukuje się dla niej nieużywany blok pamięci. Dzięki temu minimalizuje się liczbę transferów nakładek z pamięci dyskowej.

REALIZACJA SZYBKICH EKSTRAKODÓW

Stała część egzekutora realizuje wszystkie szybkie ekstrakody ODRY, a mianowicie: SUSBY, ALLOT (sprawdzanie stanu urządzenia), PERI (transfery), SUSMA, AUTO, SUSAR, SUSIN, GIVE (oprócz N (M) = 4) oraz RRQ.

Utrzymywany jest format i postać danych komputera ODRA na wszystkich wymienialnych między instalacjami R-32 i ODRY nośnikach (taśmie magnetycznej, kartach dziurkowanych oraz na wydrukach). W tym celu podczas transferów przeprowadzana jest odpowiednia konwersja.

Wskutek zasadniczych różnic sprzętowych, bezpośrednie przenoszenie danych zapisanych na pakietach dyskowych nie jest możliwe. Możliwe jest natomiast skorzystanie z pośrednictwa taśmy magnetycznej, obsługiwanej przez standardowy program pomocniczy ≠XPJW. Rys. 3 zawiera schemat obrazujący funkcje emulatora.



Rys. 3. Schemat funkcjonalny emulatora

PROCEDURY OBSŁUGI PAMIĘCI DYSKOWEJ

Spośród zrealizowanych w części stałej procedur transferów, stosunkowo najbardziej skomplikowane są procedury obsługi pamięci dyskowej.

Obsługiwane przez egzekutor E6RM i najczęściej spotykane w instalacjach ODRY pamięci dyskowe EDS 8 i EDS 30/60 mają fizyczną organizację sektorową. Charakterystyczny dla tego typu organizacji jest sztywny podział ścieżki na pewną liczbę równej wielkości fragmentów, zwanych blokami lub sektorami. Położenie bloku określają trzy współrzędne: jego numer, numer głowicy i numer cylindra. Jeden blok mieści 128 słów danych, a na jednej ścieżce EDS 8 znajduje się 8 bloków (w EDS 30/60 — 15 bloków).

Programowy dostęp do zbioru dyskowego, realizowany przez E6RM, polega na zamianie adresu logicznego porcji, występującego w programie (w polu sterującym instrukcji PERI), na adres fizyczny bloku oraz polecenie wykonawcze, zrozumiałe przez sprzęt pamięci dyskowej. Algorytm zamiany adresu wykorzystuje w tym celu m. in. specyficzne dla danego typu pamięci dyskowej dane, takie jak liczba bloków na jeden cylinder, liczba ścieżek w cylindrze oraz liczba cylindrów, przechowywane w nagłówku sterującym pakietu dyskowego.

W przypadku pamięci EDS 30/60 całość funkcji związanych z fizyczną komunikacją z pamięcią dyskową realizuje inteligentny sterownik PF56, będący w istocie minikomputerem 16-bitowym. Pamięć EDS 8 obsługiwana jest bezpośrednio przez stosowne procedury egzekutora E6RM.

Dzięki ustalonej wielkości bloków oraz przyjętemu sposobowi adresacji w programowej instrukcji PERI, istnieje znaczna niezależność między logiczną strukturą zbioru a jego postacią fizyczną.

[2]) Słowo ODRY (24 bity) jest w emulatorze przechowywane na pozycjach 0—23 32-bitowego słowa R-32; bity 24—31 są zerami

PAMIĘĆ DYSKOWA JS EMC

W odróżnieniu od pamięci dyskowej ODRY, pamięć dyskowa spotykana w instalacjach R-32 (JS 5552 i JS 5561) ma całkowicie inną organizację zapisu danych na ścieżce. Nie ma tu ustalonych sektorów, natomiast istnieje możliwość tworzenia zmiennej liczby oddzielnie i różnorodnie adresowanych zapisów o zmiennej długości. Liczbę i sumaryczną wielkość zapisów ogranicza tylko pojemność ścieżki. Każdy zapis składa się z pola identyfikującego jego numer na ścieżce, opcjonalnego pola zawierającego klucz oraz z pola danych.

Istnieje wiele operacji, zrealizowanych sprzętowo, umożliwiających dostęp do poszczególnych pól zapisu lub całego zapisu, zarówno według jego numeru na ścieżce, jak i zawartości pola klucza. Pełny adres oczywiście zawiera również numer głowicy i numer cylindra [1].

Procedura komunikacji z pamięciami dyskowymi JS 5552 i JS 5561, działającymi w kanale selektorowym R-32, jest stosunkowo najbardziej zbliżona do procedury używanej przez E6RM we współpracy z pamięciami EDS 8.

REALIZACJA WSPÓŁPRACY EGZEKUTORA Z DYSKAMI JS EMC

Wskutek wspomnianego podobieństwa, podjęto decyzję zamodelowania na dyskach JS 5552 i JS 5561 struktury fizycznej dysków EDS 8, tzn. zarówno sposobu adresacji, jak i repertuaru poleceń wykonawczych.

Pamięć EDS 8 realizuje siedem następujących poleceń:

- czytaj jeden lub więcej bloków w ramach jednego cylindra
- pisz jeden lub więcej bloków w ramach jednego cylindra
- czytaj bez transferu danych jeden lub więcej bloków w ramach jednego cylindra
- wybierz określony cylinder
- pisz identyfikatory sektorów w ramach ścieżki
- pisz z odczytem kontrolnym jeden lub więcej bloków w ramach ścieżki
- skasuj błędny sektor [5].

Jedynie polecenie „wybierz określony cylinder" ma bezpośredni odpowiednik w JS 5552 i JS 5561. Realizacja pozostałych poleceń wymaga kompilacji odpowiednio łańcuchowanych programów kanałowych, specyficznych dla każdego transferu i jest dosyć skomplikowana.

Struktura pseudosektorowa ścieżki tworzona jest poleceniem „pisz identyfikatory". Polecenie to, stosowane podczas inicjowania nowych pakietów EDS, tworzy identyfikatory sektorów, zawierające ich numery używane przy adresacji. W przypadku pamięci JS 5552 i JS 5561 powoduje utworzenie sześciu lub jedenastu zapisów na ścieżce, zawierających zarówno w polu identyfikatora, jak i w polu klucza ten sam numer zapisu oraz 512-bajtowe (128-słowowe) puste pole danych.

Polecenia czytania i pisania odszukują symulowany sektor według numeru, korzystając z pola klucza. Jeżeli dowolny zapis danych ulegnie uszkodzeniu, można go usunąć poleceniem „skasuj błędny sektor". Polecenie to odszukuje właściwy sektor również według numeru, lecz korzystając z pola identyfikatora, a następnie kasuje pole klucza, dzięki czemu kolejne próby dostępu do tego bloku będą zawsze zakończone niepowodzeniem. Mechanizm ten jest wykorzystywany w procedurze przydzielania bloków zastępczych w przypadku błędów zapisu oraz wyszukiwania bloków zastępczych przy błędach odczytu (ang. FLAW ACTION).

INICJOWANIE PAKIETÓW DYSKOWYCH

W wyniku symulacji struktury sektorowej pakiety dyskowe JS EMC uzyskały następujące parametry:

- JS 5552: 6 sektorów na ścieżkę, 10 ścieżek w cylindrze, 202 cylindry
- JS 5561: 11 sektorów na ścieżkę, 20 ścieżek w cylindrze, 202 cylindry.

Tego typu parametrów nie przewidywał standardowy program ODRY ≠XPJ1 do inicjowania dysków. Stąd też jako jedyny spośród programów ODRY nie może być stosowany pod emulatorem. Zamiast niego stosuje się specjalnie napisany program ≠SPJ1 [8]), realizujący wszelkie funkcje ≠XPJ1 i sterowany analogicznymi parametrami.

CHARAKTERYSTYKA LICZBOWA

Stała część egzekutora, jak również pozostałe moduły zaznaczone na rys. 2, zostały napisane w języku asemblera JS EMC i składają się z ok. 8 tys. linii kodu. Interpreter zawiera ok. 2 tys. linii. Sumarycznie część stała zajmuje ok. 50 KB pamięci operacyjnej (ok. 12 K słów).

Nakładki, w łącznej liczbie ok. 90, zajmują ok. 25 tys. linii w języku asemblera ODRY, co odpowiada ok. 22 K słów skomplikowanego kodu.

E6RM a OS/JS

Porównując sposób, w jaki obsługiwane są jednostki pamięci taśmowej oraz dyskowej, działające pod emulatorem i systemem operacyjnym OS/JS, nie można oprzeć się zdziwieniu, że tak rozległy system, jakim jest OS, w tak prymitywny sposób nimi zarządza. Egzekutor E6RM w każdej chwili panuje nad tym, które jednostki pamięci dyskowej i taśmowej są sprawne i jakie zawierają woluminy. Zarówno założenie, jak zdjęcie woluminu jest natychmiast rozpoznawane i zapamiętywane. Dlatego też działając pod E6RM nie ma konieczności przenoszenia tego samego woluminu w czasie przetwarzania z jednej jednostki na inną.

Podobnie system OS/JS nie radzi sobie z problemem fragmentacji pamięci operacyjnej oraz mniej efektywnie zarządza zbiorami dyskowymi. Gwoli sprawiedliwości należy jednak stwierdzić, że tak OS jak i DOS mają lepsze od E6RM procedury obsługi błędów na taśmie magnetycznej.

---

[8]) Program ≠SPJ1 jest programem zaufanym klasy S; jego autorem jest mgr Andrzej Czerwiński z ZETO Gdańsk

LITERATURA

[1] DSO IZOT: JS 5561 Techniczeskoje Opisanije. (U13.057.010 TO)

[2] Dziedzic M., Perycz K., Wiliński J.: Wieloprogramowy dyskowy symulator ODRY 1305 na R-32. INFORMATYKA 7—8, 1983

[3] Dziedzic M., Perycz K., Wiliński J.: Emulator E6RM-E — przykład wykorzystania komputerów mikroprogramowanych. INFORMATYKA 2, 1985

[4] ICL 1900 Series: E6RM Overlaid Executive Manual, Part 1,2. (IM103)

[5] ICT 11900 Series: Types 2801-2 EDS. (ID5016822)

[6] MERA-ELWRO: Architektura Logiczna m.c. ODRA 1305 — DT-R Jednostka Centralna, Tom IX (AL — 1130602-8)

[7] MERA-ELWRO: Egzekutor E6RM dla m.c. ODRA 1304 i ODRA 1305. (137905)

[8] MERA-ELWRO: Struktura Logiczna m.c. R-32 DT-R, Wyd. 3. (OL-2045901-2)

Ustąpiwszy na trzy miesiące miejsca mikrokomputerowemu przebojowi, jakim jest LOGO, w dziale „Języki programowania i systemy operacyjne" wracamy do cyklu zapoczątkowanego językiem BCPL, a mającego zakończyć się opisem UNIXA. Temat, który podejmujemy w tym numerze INFORMATYKI, sięga roku 1975. Należy przyznać, że podejmujemy go z wahaniem — głównie dlatego, że nie wiemy, czy na hasło „RATFOR" Czytelnicy nasi odpowiedzą: „RATFOR? A tak, znamy, znamy...", czy też: „...a co to takiego?"

W listopadzie 1975 r., a więc w czasach, gdy idee programowania strukturalnego rozbrzmiewały już donośnie, a jednocześnie ich przymiarki do FORTRANU były raczej przypadkowe i kończyły się nie najlepszym rezultatem — FORTRAN 77 miał wtedy minus dwa lata — w czasopiśmie SOFTWARE PRACTICE and EXPERIENCE artykuł pióra B. W. Kernighana o wymownym tytule: „RATFOR — a preprocessor for a rational FORTRAN" [3].

O ile, co do upowszechnienia nazwy RATFOR można mieć pewne wątpliwości, to nazwisko samego twórcy — zwłaszcza po latach — przydaje systemowi dużego blasku. Nie jest to jednak jedyna przyczyna, dla której wypada mówić RATFOR (to od razu po języku C); są też dwa inne powody:

- RATFOR przebył bez szwanku próbę czasu, jest wciąż (i na nowo) używany i stosunkowo łatwo dostępny — także w kraju
- RATFOR umożliwia przenośność oprogramowania, a w szczególności zapewnia mobilność języka ICON, który przedstawimy w następnej kolejności.

ZDZISŁAW PŁOSKI
Instytut Informatyki
Uniwersytet Wrocławski

RATFOR — czyli strukturalny FORTRAN (I)

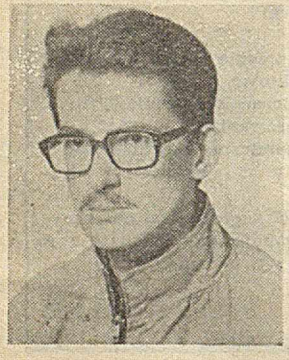
RATFOR tworzy na bazie FORTRANU swobodnoformatowy język wyposażony w konstrukcje strukturalne. Odnajdujemy wśród nich klasyczne instrukcje w rodzaju: instrukcji warunkowej, instrukcji pętli REPEAT, WHILE, specyficznie rozwiązaną instrukcję wariantową SWITCH oraz w szczególny sposób rozbudowaną instrukcję pętli FOR. Stosowanie pętli wzbogacają konstrukcje przerywnikowe — zawoalowane skoki.

Możliwości nazywania stałych (ogólnej — definiowania dowolnych ciągów znaków — napisów) i dołączania plików zewnętrznych powiększają komfort „wysławiania się" w RATFORZE. Dochodzą do tego drobne, ale cenne ulepszenia leksykalne — „cukier syntaktyczny", jak to określił jeden z krytyków [1]. Do ulepszeń tych należy zaliczyć przejrzyste operatory, możliwość posługiwania się napisami bez deklarowania ich długości, nawiasy do zaznaczania podporządkowań strukturalnych w ciągu instrukcji. Można jawnie zaznaczać instrukcje FORTRANU wyjęte spod przetwarzania i na odwrót — wszystko, co nie stanowi wyróżnionej konstrukcji RATFORU jest przekształcane do standardowego formatu FORTRANU, z podziałem na pole etykiety i pole instrukcji.

INSTRUKCJE STRUKTURALNE

RATFOR interpretuje następujące instrukcje organizujące przebieg sterowania w programie [1]):

```
IF (warunek) instrukcja
IF (warunek) instrukcja ELSE instrukcja
SWITCH (wyrażenie) {[CASE wykaz numerów wariantów: instrukcja]+ DEFAULT: instrukcja}
WHILE (warunek) instrukcja
REPEAT instrukcja
REPEAT instrukcja UNTIL (warunek)
FOR (zapoczątkowanie; warunek; wyrażenie sterujące) instrukcja
FOR ({[instrukcja]+}; warunek; {[instrukcja]+}) instrukcja
DO wykaz
BREAK [numer].
NEXT [numer].
RETURN [(wyrażenie)].
{[instrukcja]+}, albo [[instrukcja]+], albo $([instrukcja]+$)
liczba instrukcja
% instrukcja fortranowska
wszystko inne
```



Mgr ZDZISŁAW PŁOSKI ukończył w 1976 r. studia matematyczne w Uniwersytecie Wrocławskim. Pracuje w Instytucie Informatyki Uniwersytetu Wrocławskiego. Praktykujący programista, przejawia zainteresowania metodami przetwarzania tekstów. Popularyzuje język i projekt ICON. Hobby — mikrokomputery.

Pod określeniem **instrukcja** należy rozumieć dowolną z wymienionych instrukcji strukturalnych, każdą instrukcję fortranowską (z dopuszczeniem swobodnego formatu ulokowania jej elementów) oraz ciąg tychże, ujęty w nawiasy klamrowe { } lub identycznie interpretowane nawiasy [ ] albo $($). Ciąg instrukcji RATFORU, ujęty w nawiasy, stanowi strukturalnie wyodrębnioną (podporządkowaną) całość — możemy go utożsamić z klasycznie pojmowaną instrukcją złożoną.

Wiersze zaczynające się od znaku % są kopiowane do wyjściowego pliku RATFORU bez zmian, wyjąwszy usunięcie znaku ostrzegawczego (%). Winny zatem zachowywać reguły składni i formatu FORTRANU.

Ogranicznikiem instrukcji jest koniec wiersza (nl), jeśli dotychczasowy ciąg symboli określa kompletną instrukcję, lub — średnik. W razie potrzeby instrukcję można kontynuować w nowym wierszu, pozostawiając jako ostatni znak w wierszu poprzednim przecinek lub któryś z operatorów: +, —, *. Niezamknięcie nawiasów ( ) też wymusza interpretację następnego wiersza, jako dalszego ciągu nieskompletowanej instrukcji. Kontynuowanie instrukcji można także zaznaczyć jawnie umieszczając jako ostatni w wierszu znak podkreślenia (_).

---

[1]) Powtórzenia w składni instrukcji, ze względu na czytelność nawiasów, stanowiących elementy języka, zaznaczamy za pomocą nawiasów [ ]+ lub [ ]*, równoważnych odpowiednio { }+ i { }* w konwencji EBNF

*) Tytuł artykułu został sformułowany przez Redakcję.

Należy zaznaczyć, że słowa kluczowe (pisane dużymi literami) są w realizacjach RATFORU zastrzeżone — nie stosuje się żadnych oznaczników wyodrębniających je w tekście.

KONWENCJE WYRAŻANIA NAPISÓW

Wyrażenia

```
STRING nazwa „ciąg znaków"
```

lub

```
STRING nazwa (rozmiar) „ciąg znaków"
```

powodują zadeklarowanie całkowitoliczbowej tablicy fortranowskiej, wystarczająco długiej (w pierwszym przypadku) dla zapamiętania podanego ciągu znaków (napisu) lub ciągu o podanym rozmiarze. Znaki ciągu są pamiętane po jednym w słowie. Na końcu każdego ciągu znaków zapamiętuje się symbol końcowy (EOS), określony przez użytkownika. Jeśli występuje kilka deklaracji STRING jedna po drugiej, to procesor RATFORU najpierw tworzy opisy tablic, a potem je inicjuje za pomocą fortranowskiej instrukcji DATA.

RATFOR bazuje na kodzie ASCII. Poszczególne realizacje w modułach we-wy muszą zawierać tablice konwersji kodów znaków. Znaki specjalne ASCII — nowy wiersz, tabulacja itd. mogą być zapamiętywane w napisie. Oznacza się je symbolicznie dwuznakami ₒn, ₒt itd.

W parametrach procedur można odwoływać się do napisów przez przypisane im nazwy lub używać wprost stałych napisowych w postaci takiej, jak w wyrażeniach deklarujących, np.:

```
CALL BADERR („FATAL ERROR IN FOR STATEMENT PARSING.")
```

Dopuszcza się ujmowanie stałych napisowych w apostrofy '...'. Jeśli napis nie mieści się w jednym wierszu, można go kontynuować w wierszu następnym, zaznaczając przeniesienie przez zakończenie niekompletnego wiersza znakiem podkreślenia (_). Wówczas w następnym wierszu pominięte zostaną początkowe spacje i znaki tabulacji, co umożliwia dostosowanie układu graficznego zapisu do wymogów czytelności (stosowanie wcięć).

DEFINIOWANIE NAZW

Ważną konstrukcją RATFORU jest wyrażenie DEFINE, umożliwiające określenie nazwą pewnego ciągu znaków. Zwykle ciąg taki oznacza liczbę w FORTRANIE. Ogólnie rzecz biorąc, konstrukcja DEFINE stanowi w RATFORZE najprościej pojęty środek makrogeneracji. Ma ona postać:

```
DEFINE (nazwa, wartość)
```

gdzie **nazwa** musi zaczynać się od litery i może zawierać znaki alfanumeryczne oraz kropki i podkreślenia (rozróżnia się tu litery małe i wielkie). Składni **wartości** nie precyzuje się (poza oczywistymi uwarunkowaniami syntaktycznymi konstrukcji DEFINE). Maksymalna długość tekstu wartości wynosi 200 znaków.

Operując terminologią makrogeneratorów, która jest tu w pełni adekwatna, można powiedzieć, że konstrukcja DEFINE stanowi makrodefinicję, parametr **nazwa** — nazwę makrowywołania, **wartość** — tekst zastępujący (wzorzec tekstu wynikowego), a późniejsze wystąpienia zdefiniowanej nazwy w tekście przetwarzanym przez preprocesor RATFORU rozumie się jako makrowywołania (generujące określony tekst wynikowy).

Technika wykorzystania konstrukcji DEFINE w RATFORZE — poza oczywistą, wynikającą z zasad makrogeneracji — polega na tym, że każdorazowo przed przetworzeniem określonego pliku preprocesor RATFORU odczytuje plik „standardowych definicji", mianujących najczęściej używane symbole, parametry ilościowe przebiegu itp. Przykładowo, w przypadku pierwotnego tłumaczenia przez preprocesor RATFORU plik standardowych definicji liczy 135 elementów, z których dla ilustracji zacytujemy jedną:

```
DEFINE (CHARACTER, INTEGER)
```

DOŁĄCZANIE PLIKÓW I INNE ELEMENTY JĘZYKA

Konstrukcja INCLUDE, umożliwiająca dołączenie plików jest w oczywisty sposób — jak cały pakiet we-wy — bardzo zależna od konkretnej instalacji. Jej opis zewnętrzny jest następujący:

```
INCLUDE „nazwa pliku"
```

Wystąpienie INCLUDE w tekście przetwarzanym przez preprocesor RATFORU powoduje podanie na wejście zamiast dotychczas odczytywanego pliku — pliku o nazwie wskazanej w cudzysłowie. Cudzysłów zezwala na użycie nazwy składającej się z więcej niż jednego ratforowego symbolu, np. nazwy pliku z rozszerzeniem dla komputerów SM-4. Cudzysłów można opuścić, gdy nazwa ma składnię fortranowską.

Typowym zastosowaniem INCLUDE jest odczytywanie bloku wspólnych deklaracji na początku opisu różnych procedur.

Po wyczerpaniu zawartości pliku dołączonego przez INCLUDE preprocesor RATFORU automatycznie przywraca poprzedni punkt czytania. W realizacji [4] można w ten sposób utworzyć stos do pięciu plików otwartych do czytania.

Poza wymienionymi instrukcjami strukturalnymi oraz deklaracjami napisów, nazw i manipulatorem plików preprocesor interpretuje jeszcze instrukcje z etykietami. Oznacza to możliwość użycia fortranowskiej etykiety przed dowolną instrukcją (ratforową lub fortranowską), przy czym gwarantuje się niezbędną dla FORTRANU tabulację.

Tekst na prawo od znaku # aż do końca wiersza jest traktowany jako komentarz. Literalną interpretację wiersza zapewnia rozpoczęcie go od znaku %. Każdy inny wiersz zostanie potraktowany w ten sposób, że pierwszy widoczny w nim znak znajdzie się w siódmej kolumnie (przesunięcie związane z tabulacją instrukcji w FORTRANIE) i nic poza tym nie będzie zbadane, a ewentualne błędy w składni wykryje dopiero kompilator FORTRANU.

ZAPIS OPERATORÓW I LICZB

Fortranowskie operatory relacji arytmetycznych i operatory logiczne znajdują w RATFORZE następującą transkrypcję [2]):

.LT. → <
.LE. → <=
.EQ. → ==
.NE. → != albo ^= albo ~=
.GE. → >=
.GT. → >
.OR. → |
.AND. → &
.NOT. → ! albo V^ albo ~.

Dowolny znak ASCII ujęty w apostrofy jest transformowany na liczbę stanowiącą jego kod, np. 'A' jest tłumaczone na 66. Można w ten sposób używać też kodów znaków specjalnych, pisząc mnemonicznie 'ₒn' dla nowej linii, 'ₒt'— dla tabulacji itd.

W RATFORZE rozszerzono składnię liczb całkowitych o możliwość używania dowolnych podstaw. Zapis

```
n%ddd...
```

wyrażający podstawę systemu liczbowego **n** i ciąg cyfr w tym systemie **ddd...** pozwala przedstawić liczby w dowolnym praktykowanym systemie. Dla cyfr liczb szesnastkowych dopuszcza się konwencjonalne oznaczenia literowe, np.

```
8%77    2%0010011    16%2F
```

PRZYKŁADY

Poniżej dokonano przeglądu zasygnalizowanych konstrukcji, stosując transkrypcję operatorów i nawiasów, przyjętą na ODRZE 1305. W szczególności użyto nawiasów [ ] w roli { }, tj. nawiasów grupujących instrukcje. Z wyjątkiem ilustracji instrukcji wariantowej wszystkie przykłady pochodzą z wersji RATFORU opisanej w [2].

1) Instrukcja warunkowa jednoczłonowa

```
IF (I > DEFSIZ)
  CALL BADERR („DEFINITION TOO LONG.")
```

---

[2]) Na ODRZE 1305 zastosowano transkrypcję .NE. → £= oraz .OR. → ↑

2) Instrukcja warunkowa z członem ELSE

```
IF (C == LPAREN)
   T = LPAREN      # DEFINE NAME, DEFN
ELSE [
   T = BLANK       # DEFINE NAME DEFN [3])
   CALL PBSTR (PTOKEN)
```

Człon ELSE zawiera tu instrukcję złożoną, ujętą w nawiasy [ ].

3) Kaskada instrukcji warunkowych

```
# TYPE — RETURN LETTER, DIGIT OR CHARACTER;
# WORKS WITH ASCII ALPHABET
INTEGER FUNCTION TYPE (C)
INTEGER C
IF (C >= DIG0 & C <= DIG9)
   TYPE = DIGIT
ELSE IF (C >= LETA & C <= LETZ)
   TYPE = LETTER
ELSE IF (C >= BIGA & C <= BIGZ)
   TYPE = LETTER
ELSE
   TYPE = C
RETURN
END
```

W przykładzie tym warto zwrócić uwagę na wmontowanie struktury w opis funkcji fortranowskiej i na swobodny format zapisu instrukcji fortranowskich.

4) Instrukcja wariantowa

Jest to druga co do złożoności — obok instrukcji FOR — konstrukcja RATFORU, choć rzadko używana.

```
SWITCH (ARG (J)) [
   CASE 3:
      IF (MFLAG == NO)
         CFLAG = YES
   CASE 13, 15, 17—20:
      MFLAG = YES
         CFLAG = NO
   CASE 25:
      ...
   DEFAULT:
      CALL USAGE
      ]
```

Warianty są oznaczane numerami, przy czym jeden wariant może dotyczyć całego wykazu oznaczeń. Wykaz tworzą poszczególne numery lub ich przedziały — zaznaczane „od-do". W wypadku większej liczby oznaczeń wariantu, gdy łącznie rozpatrywany przedział numerów jest wypełniony powyżej 50%, odpowiedni fragment tłumaczy się na fortranowskie instrukcje GOTO.

Jeśli żaden z numerów podanych w wykazach CASE nie odpowiada wartości wyrażenia sterującego (podanego w nawiasach po SWITCH), zostaje wykonany wariant domyślny (DEFAULT). Wszystkie warianty zamknięte są w nawiasy [ ]. Zauważmy, że w poszczególnych wariantach można pisać ciągi instrukcji bez konieczności zamykania ich w nawiasy [ ].

5) Pętle WHILE i REPEAT

```
# OUTTAB — GET PAST COLUMN 6
   SUBROUTINE OUTTAB
   INCLUDE COUTLN
   WHILE (OUTP < 6)
      CALL OUTCH (BLANK)
   RETURN
   END
```

W powyższym przykładzie wnętrze instrukcji WHILE stanowi jedna tylko instrukcja (CALL). Chcąc podporządkować strukturze więcej instrukcji, ujmuje się je w nawiasy [ ]. W opisie procedury widoczna jest wstawka pliku COUTLN (konstrukcja INCLUDE).

Warunek sterujący iteracją może być dowolnie złożony, na przykład w instrukcji:

```
WHILE (C == BLANK ↑ C == TAB)
   # COMPRESS MANY BLANKS TO ONE
   C = NGETCH (C, FD)
```

[3]) Komentarze dotyczą istnienia nieudokumentowanego oficjalnie wariantu konstrukcji DEFINE

stanowi on alternatywę dwóch relacji równości.

Wnętrze pętli może być puste, co zaznacza się następująco:

```
WHILE (NGETCH (C, FD), £ = NEWLINE) # STRIP COMMENTS
   ;
```

Przykład pętli REPEAT jest następujący:

```
REPEAT [
   T = GNBTOK (TOKEN, MAXTOK)
   CALL OUTSTR (TOKEN)
] UNTIL (T == SLASH)
```

Rozważmy instrukcję

```
REPEAT [
   T = GETTOK (TOKEN, MXTOK)
   IF (T == SEMICOL)
      BREAK
   IF (T == NEWLINE & NLPAR == 0)
      BREAK
   IF ...
   ...
]
```

Tak zorganizowana pętla tworzy zarys nie kończącej się iteracji. Efekt braku warunku symuluje się czasem w innych językach warunkiem zawsze fałszywym. Zamknięciu pętli (opuszczeniu jej) służą instrukcje BREAK [4]).

[4]) Modne od lat programowanie bez skoków w podobny sposób ukrywa instrukcję GOTO również w innych językach

LITERATURA

[1] Computing Reviews, Vol. 17, No 6, June 1976, notka redakcyjna nr 29939

[2] Hanson D. R.: Installing Version 3 of the Software Tools. Report TR 81—23, Dept. of Computer Science, The University of Arizona, Tucson, 1981

[3] Kernighan B. W.: RATFOR — A Preprocessor for a Rational FORTRAN. Software — Practice and Experience, Vol. 5, pp. 395—406, October 1975

[4] Software Tools Distribution — Version 3. Dept. of Computer Science, The University of Arizona, Tucson, January 1982.

CONVENTION INFORMATIQUE

W dniach od 15 do 19 września 1986 r. w paryskim Pałacu Kongresów już po raz siedemnasty obradować będzie międzynarodowa konferencja CONVENTION INFORMATIQUE. W wyniku konsekwentnego wiązania z wystawą SICOB, impreza ta przekształciła się w największą w Europie cykliczną międzynarodową konferencję informatyczną (w 1985 r. ponad 3400 uczestników).

Ramowy program konferencji obejmuje:

- rynki i trendy rozwojowe (Markets and trends)
- rozwój technik i metod (Development in techniques and methods)
- zastosowania (Applications)
- mikrokomputery (Microcomputers)
- zarządzanie i aspekty ekonomiczne (Management and economic aspects).

Komitet programowy zaprasza do nadsyłania do końca 1985 r. propozycji referatów w języku angielskim lub francuskim. Powinny one zawierać: tytuł referatu, nazwisko i adres zamieszkania (telefon) autora oraz nazwę instytucji, w której jest on zatrudniony; streszczenie referatu o maksymalnej objętości 25 wierszy maszynopisu; krótki życiorys autora.

W terminie do 15 kwietnia 1986 r. autorzy otrzymają decyzję o przyjęciu propozycji, wraz ze szczegółowymi wskazówkami na temat przygotowania tekstu referatu. Tekst ten, poprzedzony streszczeniem w obu urzędowych językach konferencji, powinien być nadesłany w terminie do 27 czerwca 1986 r.

Autorom referatów nie będą zwracane koszty przejazdu i pobytu w Paryżu. Będą oni musieli opłacać uczestnictwo w sesjach innych niż ta, w ramach której wygłaszają referat. Zwłaszcza ostatnia zasada, całkowicie odmienna od dotychczasowej praktyki przytłaczającej większości konferencji, chyba najlepiej potwierdza fakt osiągnięcia przez CONVENTION INFORMATIQUE bardzo wysokiego prestiżu międzynarodowego, pozwalającego organizatorom zrezygnować ze stosowania tradycyjnych bodźców pozyskiwania autorów.

Adres dla korespondencji: Convention Informatique, 4—6, place de Valois, 75001 Paris (France), tel. 42-61-46-21, 42-61-52-42, teleks 212597 F.

BOLESŁAW MIKOŁAJCZAK
JANUSZ STOKŁOSA
Instytut Automatyki
Politechnika Poznańska

Złożoność algorytmów (2)

Złożoność asymptotyczna

Do zasadniczych cech miary dobroci algorytmów zaliczamy: zgodność ze specyfikacją, weryfikowalność, testowalność, modyfikowalność, łatwość konserwacji i niskie koszty eksploatacji. Do ważnych cech algorytmów należą: prostota, jasność i dostosowanie do spodziewanych danych [3]. W tej części artykułu zwrócimy szczególną uwagę na jedną z najważniejszych, obok poprawności, cechę podmiotową algorytmów, a mianowicie złożoność czasową i pamięciową (z których ważniejszą jest ta pierwsza).

Do badania złożoności czasowej algorytmów można podejść w różny sposób. Jednym ze sposobów byłoby jego zaprogramowanie i zmierzenie czasu wykonywania dla pewnej szczególnej implementacji na wybranym komputerze i przy dostępnych danych. Podejście to jest powszechnie stosowane, zawiera wszakże wiele słabości:
— złożoność czasowa zależy nie tylko od programowanego algorytmu, ale również od zbioru instrukcji komputera, jakości kompilatora i zręczności programisty
— implementacja może być dobrze dostrojona do pracy na specyficznym zbiorze danych, a całkowicie nieefektywna dla większości pozostałych zbiorów danych.

Czynniki te mogą zmieniać się też zależnie od wykorzystywanego komputera, kompilatora i umiejętności programisty, mogą być znacząco różne dla różnych danych. Dla zredukowania wpływu tych czynników wprowadzono **asymptotyczną złożoność czasową** jako podstawową miarę oceny działania algorytmów (w szczególności złożoność czasową według najgorszego przypadku). Jako pomocniczą miarę oceny działania algorytmów używa się przeciętnej (średniej) złożoności czasowej, dla której niezbędne jest przyjęcie pewnych założeń o rozkładzie danych wejściowych. Założenia te powinny być zbliżone do rzeczywistości, co sprawia, że problem wyznaczania przeciętnej złożoności znacznie się komplikuje. Dlatego też będziemy tu rozważać złożoność według najgorszego przypadku (złożoność pesymistyczną).

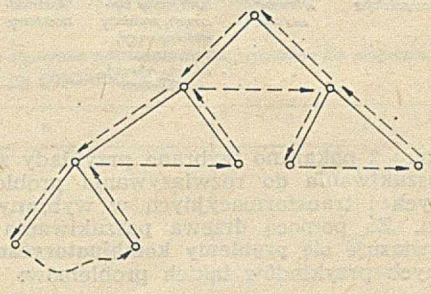
W definicji efektywności algorytmu pomija się stałe czynniki wpływające na czas obliczeń. Ma to swoje uzasadnienie, ponieważ:
— algorytmy są zapisywane zwykle w językach wysokiego poziomu
— algorytmy są opisywane za pomocą kroków, z których każdy wymaga stałego czasu niezbędnego do translacji na kod maszynowy dowolnego komputera. Ile tego stałego czasu potrzeba, nie zależy jednak tylko od samego kroku, ale również od procesu translacji i repertuaru instrukcji rozważanego komputera, a zatem wszelka próba bardziej precyzyjnego wypowiedzenia się o złożoności czasowej inaczej, że „jest rzędu f(n)" wymagałaby uwzględnienia szczegółów dotyczących tylko tej specyficznej maszyny.

Kolejnym istotnym powodem, dla którego zajmujemy się złożonością asymptotyczną i zaniedbujemy czynniki stałe, jest to, że złożoność asymptotyczna o wiele bardziej niż czynniki stałe określa dla jakiego rozmiaru wejść algorytm może być wykorzystany na komputerze. Oczywiście dla wielu ważnych problemów, jak na przykład sortowanie, możemy być zainteresowani analizą złożonościową, której rezultatem będzie wynik, że na typowym komputerze „algorytm A powinien przebiegać trzy razy szybciej niż algorytm B". Ponadto w sytuacji, jeśli znamy spodziewany rozkład prawdopodobieństwa danych wejściowych, trafniejsza jest analiza przeciętnej złożoności obliczeniowej.

Poniżej przedstawiamy sposób praktycznego podejścia do problemów trudnych obliczeniowo przy użyciu algorytmu nawrotu, który zostanie zaprezentowany za pomocą znanego problemu kolorowania mapy. Problemy łatwe obliczeniowo, a więc takie, że ich złożoność czasowa jest wielomianową funkcją długości wejścia, zostaną tu omówione na przykładzie problemu sortowania liczb generowanych przez generator liczb pseudolosowych, z zastosowaniem algorytmu Shella.

PROBLEMY TRUDNE — ALGORYTM NAWROTU

Klasa NP jest bardzo szeroka. Mówiąc nieprecyzyjnie, problem decyzyjny należy do klasy NP wtedy i tylko wtedy, gdy może być rozwiązany poprzez **poszukiwanie z nawrotem**, o wysokości drzewa poszukiwania ograniczonej wielomianowo (rys. 1). Algorytm niedeterministyczny może



Rys. 1.

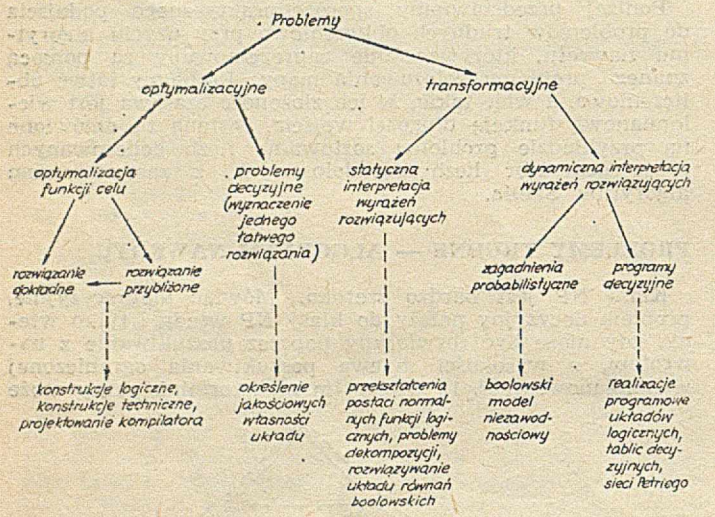
przebiegać przez wiele ścieżek obliczeniowych na drzewie poszukiwania. Liczba ścieżek obliczeniowych może rosnąć wykładniczo jako funkcja wejścia. Jednakże każda szczególna ścieżka obliczeniowa kończy się po **n** interacjach odpowiedzią pozytywną albo negatywną (gdzie **n** jest miarą wejścia). Możemy zatem uważać algorytm niedeterministyczny za proporcjonalny w czasie do **n** w tym sensie, że

Doc. dr hab. inż. **BOLESŁAW MIKOŁAJCZAK** w latach 1970—1985 pracował w Instytucie Automatyki Politechniki Poznańskiej, a obecnie pracuje w Ośrodku Informatyki tej uczelni. Zajmuje się modelami matematycznymi systemów cyfrowych, ich zastosowaniem do projektowania i eksploatacji systemów cyfrowych, złożonością obliczeniową algorytmów ze szczególnym uwzględnieniem zagadnień projektowania logicznego systemów cyfrowych.

Dr inż. **JANUSZ STOKŁOSA** w latach 1972—1985 pracował w Instytucie Automatyki Politechniki Poznańskiej. Obecnie jest adiunktem w Ośrodku Informatyki tej uczelni. Zainteresowania naukowe dotyczą dyskretnych systemów dynamicznych, złożoności algorytmów i ochrony informacji.

w ocenie złożoności czasowej tego algorytmu uwzględniamy tylko etap sprawdzania dla uprzednio wybranego niedeterministycznie „rozwiązania" in spe. Nie oznacza to oczywiście, że można zbudować program działający w czasie proporcjonalnym do n. Aby to zrobić, trzeba mieć albo wyrocznię, która wskazywałaby poprawną gałąź w drzewie poszukiwania w każdym punkcie zawierającym wybór, albo nieograniczoną możliwość wykonywania obliczeń równoległych, celem wykonywania wszystkich ścieżek obliczeniowych równocześnie (współbieżnie).

Naturalnym sposobem zamiany obliczenia niedeterministycznego w deterministyczne jest generowanie obydwu rozwiązań częściowych przy każdej sytuacji wyboru, z których jedno jest umieszczane na stosie, celem umożliwienia rozpatrzenia go w terminie późniejszym, a drugie uczestniczy w dalszym procesie obliczeniowym. Jeśli w kontynuowanym obliczeniu osiągniemy rozwiązanie negatywne, to rozwiązanie częściowe umieszczone uprzednio na stosie jest pobierane i wykorzystane w obliczeniach. Obliczenie zostanie zakończone, jeśli uzyskuje się rozwiązanie pozytywne lub jeśli stos jest pusty, co oznacza wykonanie obliczeń wzdłuż wszystkich możliwych ścieżek. Takie postępowanie nazywa się **algorytmem nawrotu.** Oczywiście czas wykonywania algorytmu nawrotu jest określony przez całkowitą liczbę rozwiązań częściowych, generowanych wzdłuż wszystkich ścieżek obliczeniowych. Liczba tych rozwiązań rośnie wykładniczo. Łatwo zauważyć, że algorytm nawrotu jest związany z podziałem problemu złożonego na dwa rozłączne podproblemy. Kolejne podziały problemu prowadzą do powstania **drzewa poszukiwania.**



Rys. 2.

Na rysunku 2 pokazano wybrane przykłady zastosowania drzewa poszukiwania do rozwiązywania problemów optymalizacyjnych i transformacyjnych w wybranych działach informatyki. Za pomocą drzewa poszukiwania szczególnie dobrze rozwiązuje się **problemy kombinatoryczne.** Oto kilka wybranych przykładów takich problemów:

- czy pewne zadanie obliczeniowe może być wykonane przez zadany a priori system cyfrowy poprzez odpowiednie podporządkowanie występujących w tym zadaniu zmiennych zewnętrznych zmiennym wewnętrznym, opisującym rozważany system cyfrowy?
- znajdowanie optymalnego rozmieszczenia elementów systemu cyfrowego na obwodzie drukowanym
- wyznaczanie minimalnych zbiorów testów wykrywających i lokalizujących uszkodzenia w systemie cyfrowym.

Oto ogólne przesłanki metodologiczne, pomocne przy konstruowaniu algorytmów z nawrotem na drzewach poszukiwania.

Na początku konstruowania drzewa poszukiwania dana jest **przestrzeń poszukiwania S**, którą można odpowiednio ukształtować przez:

- wprowadzenie zmiennych problemowych jednego typu x_i, gdzie $i = 1, 2, ..., n$ takich, że $S = Q^n$
- wprowadzenie zmiennych problemowych różnych typów i tym samym rozbicie przestrzeni poszukiwania na podprzestrzenie.

Przestrzeń poszukiwania **S** może być podzieloną na podprzestrzenie na różne sposoby:

- symetrycznie, według zmiennych rozdzielających x_i
- niesymetrycznie:
  — za pomocą płaszczyzny tnącej,
  — za pomocą równości lub nierówności dwu lub więcej zmiennych.

Ważnym elementem jest wprowadzenie systemu ograniczeń $R_i(\vec{x})$ dla $i = 1, ..., m$, nakładanych na rozważane rozwiązania. System ten wynika na ogół bezpośrednio ze specyfikacji rozważanego problemu; wszakże przy jego formułowaniu istnieje pewna dowolność, która rzutuje znacząco na nakłady czasowe i pamięciowe przy maszynowym rozwiązywaniu problemu.

Dla systemu ograniczeń $R_i(\vec{x})$ możemy zdefiniować funkcję charakterystyczną:

$f : S \to \{0,1\}$, przy czym $f(\vec{x}) = 1$ wtedy i tylko wtedy, gdy system ograniczeń jest spełniony.

Można zatem zapisać, że $f(\vec{x}) = \prod_{i=1}^{m} R_i(\vec{x})$, gdzie **f** jest funkcją boolowską dla binarnej przestrzeni poszukiwania.

Koszty czasu i złożoność podprogramów przy wyborze zmiennych, dzielących przestrzeń na podprzestrzenie, muszą być uzasadnione znaczącą oszczędnością czasu dla całego problemu (koszty czasowe dla całego problemu mogą być mierzone liczbą kroków poszukiwania pomnożonych przez przeciętny czas obliczeń dla pojedynczego kroku).

Pierwszy czynnik, a mianowicie liczba kroków poszukiwania, może być zmniejszony poprzez:

- wykrycie możliwości bezpośredniego rozwiązania podproblemu
- wykrycie identyczności lub izomorfizmu danego podproblemu z podproblemem już rozwiązanym
- wykrycie takiej dekompozycji problemu, że całkowite rozwiązanie może być uzyskane za pomocą prostej kombinacji rozwiązań podproblemów.

Z powyższych rozważań wynikają następujące zalety algorytmów poszukiwania z nawrotem:

- wykonywanie ruchów lokalnych w przestrzeni poszukiwania
- możliwość stosowania procedur rekurencyjnych
- wykorzystanie stosu dla magazynowania danych z poprzednich podproblemów
- mniejsze wymagania co do rozmiarów pamięci, ponieważ dane dla podproblemów rozłączonych nie muszą być pamiętane równocześnie (czasem jest odwrotnie: na stosie zapamiętuje się bardzo dużo informacji w programach z rekursją, czego można uniknąć przy prostym strawersowaniu wszystkich ścieżek).

Do ewentualnych wad poszukiwania z nawrotem można zaliczyć mniejsze panowanie nad całością problemu i możliwościami jego rozwiązania, co może prowadzić do długiego „pobytu" w ślepym zaułku.

Rozważamy poniżej przykład problemu kolorowania mapy, który jest w ogólnym przypadku problemem NP-zupełnym [1, 3]. Problem polega na pokolorowaniu sąsiednich krajów na mapie, używając różnych kolorów, przy czym liczba użytych kolorów dla całej mapy powinna być minimalna. Problem ten ma również zastosowanie przy projektowaniu połączeń we wszelkich urządzeniach elektrycznych, gdzie przewody łączące bloki i mające wspólny punkt końcowy powinny być oznaczone różnymi kolorami. Jest oczywiste, że problem ten jest równoważny problemowi kolorowania grafu. Ogólne rozwiązanie problemu kolorowania mapy powinno przebiegać w dwu zasadniczych krokach:

1) wyznaczenie wszystkich maksymalnych wewnętrznie stabilnych zbiorów A_i, $i = 1, ..., m$, dla grafu $G = (V, E)$, gdzie **V** jest zbiorem wierzchołków a $E \leqslant V \times V$ jest relacją sąsiedztwa [1])

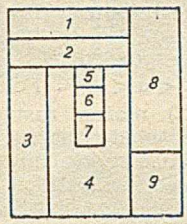
2) wyznaczenie pokrycia $\{A_{i_1}, A_{i_2}, ..., A_{i_k}\}$ zbioru wierzchołków **V** z minimalną liczbą chromatyczną $\lambda(G)$ grafu [2])

---

[1]) Przez maksymalnie wewnętrznie stabilny zbiór A_i rozumiemy taki podzbiór zbioru wierzchołków **V**, że istnieje krawędź łącząca dwa wierzchołki ze zbioru A_i i nie istnieje podzbiór A_i taki, że $A_i \subseteq A_i \subseteq V$ posiadający powyższą własność

[2]) Liczba chromatyczna $\lambda(G)$ jest najmniejszą możliwą wartością c, dla której istnieje c-kolorowanie grafu;
c-kolorowaniem grafu nazywamy podział $\Pi = \{B_1, B_2, ..., B_c\}$ jego zbioru wierzchołków, taki, że elementy należące do tego samego bloku mają różne kolorowania

3) na podstawie pokrycia wyznaczamy podział na zbiorze wierzchołków, który uzyskuje się przez usunięcie z A_{i_h} tych wierzchołków, które równocześnie występują w bloku A_{i_h}, dla $i_h \neq i_h$;



Rys. 3.

Przykład 1. Wyznaczyć minimalną liczbę kolorów dla mapy z rysunku 3. Maksymalne wewnętrznie stabilne zbiory są wyznaczone za pomocą rozwiązań boolowskich następującego zbioru ograniczeń:

$$x_i = \prod_{j=1}^{m} (\bar{a}_{ij} \vee \bar{x}_j),\ i = 1, \ldots, n$$

przy czym $a_{ij} = 1$ wtedy i tylko wtedy, gdy $(c_i, c_j) \in E$. Na podstawie rysunku 3 zbiór ograniczeń ma następującą postać:

$x_1 = \bar{x}_2\bar{x}_8$ (1) $\quad x_2 = \bar{x}_1\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_8$ (2)

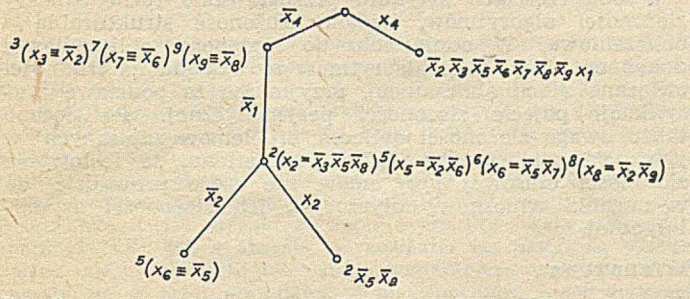
$x_3 = \bar{x}_2\bar{x}_4$ (3) $\quad x_4 = \bar{x}_2\bar{x}_3\bar{x}_5\bar{x}_6\bar{x}_7\bar{x}_8\bar{x}_9$ (4)

$x_5 = \bar{x}_2\bar{x}_4\bar{x}_6$ (5) $\quad x_6 = \bar{x}_4\bar{x}_5\bar{x}_7$ (6)

$x_7 = \bar{x}_4\bar{x}_6$ (7) $\quad x_8 = \bar{x}_1\bar{x}_2\bar{x}_4\bar{x}_9$ (8)

$x_9 = \bar{x}_4\bar{x}_8$ (9)

(np. $x_1 = \bar{x}_2\bar{x}_8$ oznacza, że kraj 1 powinien mieć kolorowanie różne od kraju 2 i kraju 8). Zbiór ograniczeń (1)÷(9) zapiszemy obecnie za pomocą jednej funkcji logicznej w postaci dysjunkcyjnej, korzystając z algorytmu nawrotu.



Rys. 4.

Jako zmiennej dzielącej rozważany problem na podproblemy użyjemy zmiennej x_4, która najczęściej ze wszystkich zmiennych występuje w równaniach (1)÷(9). Rezultaty rozważań zaprezentowano na rysunku 4, przy czym górne indeksy oznaczają numer ograniczenia, z którego korzystano. Na tej podstawie możemy wyznaczyć postać dysjunkcyjną funkcji logicznej, odpowiadającą ograniczeniom (1)÷÷(9), którą otrzymujemy wprost z drzewa z rysunku 4.

$$z = \bar{x}_4(x_3 \equiv \bar{x}_2)(x_7 \equiv \bar{x}_6)(x_9 \equiv \bar{x}_8)\bar{x}_1(\bar{x}_2(x_6 \equiv \bar{x}_5) \vee x_2(\bar{x}_5\bar{x}_8)) \vee$$
$$\vee x_4\bar{x}_2\bar{x}_3\bar{x}_5\bar{x}_6\bar{x}_7\bar{x}_8\bar{x}_9x$$

Po przekształceniach otrzymujemy:

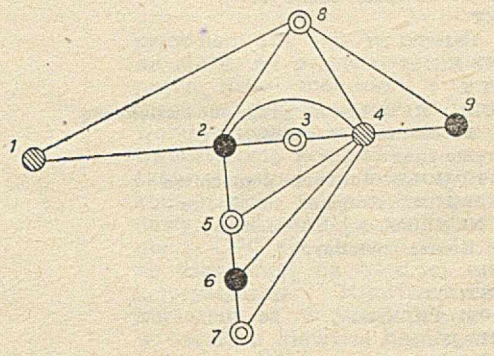
$$z = \bar{x}_1\bar{x}_2x_3\bar{x}_4(\bar{x}_5x_6\bar{x}_7 \vee x_5\bar{x}_6x_7)(\bar{x}_8x_9 \vee x_8\bar{x}_9)$$

$$\bar{x}_1x_2\bar{x}_3\bar{x}_4\bar{x}_5(\bar{x}_6x_7 \vee x_6\bar{x}_7)\bar{x}_8x_9 \vee x_1\bar{x}_2\bar{x}_3x_4\bar{x}_5\bar{x}_6\bar{x}_7\bar{x}_8\bar{x}_9$$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
A_1	0	0	1	0	0	1	0	0	1
A_2	0	0	1	0	0	1	0	1	0
A_3	0	0	1	0	1	0	1	0	1
A_4	0	0	1	0	1	0	1	1	0
A_5	0	1	0	0	0	0	1	0	1
A_6	0	1	0	0	0	1	0	0	1
A_7	1	0	0	1	0	0	0	0	0

Rys. 5.

Na rysunku 5 podano zapis tej funkcji w tabeli, w której $A_1 \div A_7$ oznaczają implikanty. Łatwo zauważyć, że implikanty A_4, A_6, A_7 tworzą minimalne pokrycie zmiennych $x_1 \div x_9$. Oznacza to, że mapa z rysunku 3 wymaga minimalnie trzech kolorów. Na rysunku 6 podano przykładowe kolorowanie grafu odpowiadającego tej mapie. Z przykładu wynika również, jak wiele problemów decyzyjnych jest ze sobą ściśle powiązanych: problem kolorowania mapy, kolorowania grafu, spełnialność funkcji logicznych, pokrycia. Ta sytuacja jest typowa. Wszystkie problemy NP-zupełne są do siebie wielomianowo przywiedlne, czyli takie, że ich wzajemne przekształcenie nie wymaga więcej czasu niż wielomianowo w funkcji wielkości wejścia [1, 5].



Rys. 6.

PROBLEMY ŁATWE OBLICZENIOWO

Problemami łatwymi obliczeniowo nazywamy problemy, które można rozwiązać w czasie wielomianowym w funkcji wielkości wejścia. W informatyce znaczną rolę odgrywa problem **sortowania** danych, który jest przykładem problemu łatwego obliczeniowo. Istnieje wiele algorytmów rozwiązujących ten problem [2]. W dalszej części rozważymy szczegółowo algorytm Shella, który ma asymptotyczną złożoność czasową równą $0(n \cdot \log n)$. Algorytm Shella zapisano w BASICU. Program te zawiera poza właściwym programem sortowania również procedurę sterującą. Sortowaniu podlegają liczby uzyskiwane z generatora liczb pseudolosowych za pomocą standardowej funkcji RND. Z testowania tego programu na komputerze osobistym COMMODORE wynika, że jego złożoność czasowa dla przebadanych rozmiarów problemu może być uważana za wprost proporcjonalną do liczby sortowanych liczb (patrz tabela); jednakże wynik testu nie upoważnia do wnioskowania odnośnie jego złożoności asymptotycznej.

Złożoność czasowa algorytmu sortowania Shella, wykonana dla programu, w BASICU, na komputerze osobistym COMMODORE 64

n	t [s]
5	1,6
10	3,2
20	6,7
40	15,3
80	34,8
160	81,9
320	196,9
640	460,2
1280	1121,8

Program sortowania Shella w języku BASIC

```
10 REM Test prędkości procedury sortowania
11 REM Zmienne
12 REM H1 HOLD
13 REM M1 MAX maksymalna długość
14 REM N1 NROW liczba wierszy
15 REM koniec zmiennych
40 M1 = 500
50 DIM X(500)
70 INPUT "Liczba danych?" ; N1
80 IF (N1<2) THEN 900
90 IF (N1>M1) THEN 70
100 FOR I=1 TO N1
110 X(I)=RND(1) : REM ewentualnie zmienić na RND(0)
120 NEXT I
130 GOSUB 600: REM Drukowanie tablic pierwotnych
150 GOSUB 700: REM Sortowanie liczb przypadkowych
170 GOSUB 600: REM Drukowanie tablic sortowanych
```

```
180 PRINT "przypadkowo"
190 GOSUB 700: REM Sortowanie tablic
210 GOSUB 600
220 PRINT "sortowany"
230 FOR I=1 TO N1
240 X(I)=N1+1-I
250 NEXT I
270 GOSUB 700: REM Sortowanie liczb zamienionych
290 GOSUB 600
300 PRINT "zamieniony"
310 GOTO 70
600 REM Drukowanie tablic X
610 PRINT
620 FOR I=1 TO N1
630 PRINT X(I)
640 NEXT I
650 RETURN: REM koniec programu wstępnego
700 REM Procedura sortowania Shella
710 REM Zmienne
711 REM H1 HOLD Zmienne pomocnicze
712 REM J6 JUMP Zmienne indeksowe
713 REM N1 NROW Liczba wierszy
714 REM Koniec zmiennych
800 J6=N1
810 J6=INT(J6/2)
820 IF(J6=0) THEN 897
830 J2=N1-J6
840 FOR J=1 TO J2
850 I=J
860 J3=I+J6
870 IF (X(I)<= X(J3)) THEN 895
890 H1=X(I)
891 X(I)=X(J3)
892 X(J3)=H1
893 I=I-J6
894 IF (I>0) THEN 860
895 NEXT J
896 GOTO 810
897 RETURN: REM Koniec sortowania Shella
900 END
```

Obecnie rozważymy sposób wyznaczania asymptotycznej złożoności czasowej **T(n)** według najgorszego przypadku dla programów rekurencyjnych. Metoda ta polega na rozwiązaniu równania różnicowego (rekurencyjnego) [3] przypisanego programowi na podstawie jego analizy. W ten sposób możemy określić złożoność czasową na podstawie analizy algorytmicznej rozważanego problemu. Przykładem będzie tu inna procedura sortowania, a mianowicie **mergesort**. Jako wejście podajemy listę o długości **n**, a jako wyjście otrzymujemy listę posortowaną o długości **n**. Procedura **mergesort** wykorzystuje ponadto podprocedurę złączania dwu list oznaczoną **merge** (L_1, L_2). Jako wejście przyjmujemy dwie posortowane listy L_1 i L_2, następnie badamy parami ich lewe skrajne elementy; większy z tych elementów jest usuwany z odpowiedniej listy i umieszczany na nowej liście jako wyjście. Wynikiem jest pojedyncza złączona lista elementów z list L_1 i L_2.

```
function mergesort (L:LIST; n:integer) :LIST;

{L jest listą o długości n. Wyjście stanowi posortowana
lista L. Przyjmujemy, że n jest potęgą liczby 2}

begin
  if n=1 then mergesort:=L
    else
      begin {podziel L na połowy L1 i L2, każda o długości
                                                    n/2}
      merge (mergesort (L1, n/2), mergesort (L2, n/2))
  end
end {mergesort}
```

Czas wymagany przez procedurę **merge** dla list o długości **n/2** jest $0(n)$. Możemy teraz zapisać ogólne równanie rekurencyjne, które wyznacza kres górny na czas **T(n)**:

$$T(n) \leqslant \begin{cases} k_1 & \text{jeśli } n = 1 \\ 2T(n/2) + k_2 n & \text{jeśli } n > 1 \end{cases}$$

Procedura mergesort zawiera więc trzy istotne części: pierwszą, służącą testowaniu czy $n \neq 1$ i wymagającą stałego czasu k_1; drugą, dzielącą listę **L** na połowy i trzecią procedurę złączania list **merge**. Części druga i trzecia procedury mergesort wymagają czasu, który jest proporcjonalny do **n**. Zatem można wybrać w taki sposób stałą k_2, aby k_2 było kresem górnym, na czas wymagany przez **mergesort** dla części drugiej i trzeciej. Ponadto w równaniu rekurencyjnym należy uwzględnić fakt przywołania rekurencyjnego, co wyrażono składnikiem $2 \cdot T(n/2)$. Z zapisu procedury **mergesort** wynika, że stosuje się ona tylko dla przypadku, gdy **n** jest potęgą liczby 2. Jednakże znając **T(n)** dla **n** będącego potęgą liczby 2, znamy również **T(n)** dla wszystkich **n**. W szczególności możemy dla wszystkich algorytmów powiedzieć, że **T(n)** leży pomiędzy $T(2^i)$ i $T(2^{i+1})$, jeśli $2^i < n < 2^{i+1}$. Ponadto dla liczb nieparzystych wyraz $2T(n/2)$ można zastąpić przez $T(n+1/2)+T(n-1/2)$ dla $n>1$ i wtedy możemy rozwiązać zmodyfikowane równanie różnicowe, celem osiągnięcia dokładnej postaci rozwiązania dla wszystkich **n**. Do rozwiązania równania różnicowego [4] stosujemy metodę „zgadywania" rozwiązania **f(n)** i następnie wykazania, że $T(n) \leqslant f(n)$. Ze szczegółowych rozważań nad rozwiązaniem równania rekurencyjnego (patrz [1]) wynika, że złożoność czasowa asymptotyczna dla problemu sortowania jest $0(n \cdot \log n)$.

Nasze dotychczasowe rozważania mówią tylko tyle, że w najgorszym przypadku rozważany algorytm wymaga $0(n \cdot \log n)$ czasu. Jeśli wybralibyśmy do rozważań funkcję **f(n)** rosnącą wolniej niż $n \cdot \log n$, to nie udałoby się udowodnić, że $T(n) \leqslant f(n)$. Kolejnym krokiem powinno być teraz wykazanie, ile wynosi przeciętna czasowa złożoność obliczeniowa, którą obliczamy [2] na podstawie znanego a'priori rozkładu prawdopodobieństwa danych wejściowych. Okazuje się, że w rozważanym przypadku przeciętna złożoność obliczeniowa jest również rzędu $n \cdot \log n$. Interesująca jest również znajomość dowolnego ograniczenia na złożoność czasową problemu. W tym wypadku zamiast symbolu $0(f(n))$ używamy $\Omega(f(n))$. Mówimy mianowicie, że funkcja **f(n)** jest $\Omega(g(n))$ (**f(n)** jest co najmniej rzędu wielkości **g(n)**), jeśli istnieje stała $c>0$ taka, że $f(n) \geqslant c \cdot g(n)$ dla prawie wszystkich wartości **n**. Okazuje się [2], że każdy algorytm sortujący za pomocą porównań wykonuje $\Omega(n \cdot \log n)$ porównań.

* * *

W obu częściach artykułu przedstawiono różne aspekty złożoności algorytmów, badano złożoność strukturalną i obliczeniową. W odniesieniu do złożoności obliczeniowej dokonano klasyfikacji algorytmów ze względu na trudności związane z ich obliczaniem, przyjmując za podstawę klasyfikacji pojęcie złożoności pesymistycznej. Poświęcono także uwagę złożoności obliczeń problemów zapisanych w języku asemblera, a ponadto podkreślono, że testowanie złożoności czasowej programów nie może prowadzić do wyciągania wniosków dotyczących ich złożoności asymptotycznej.

LITERATURA

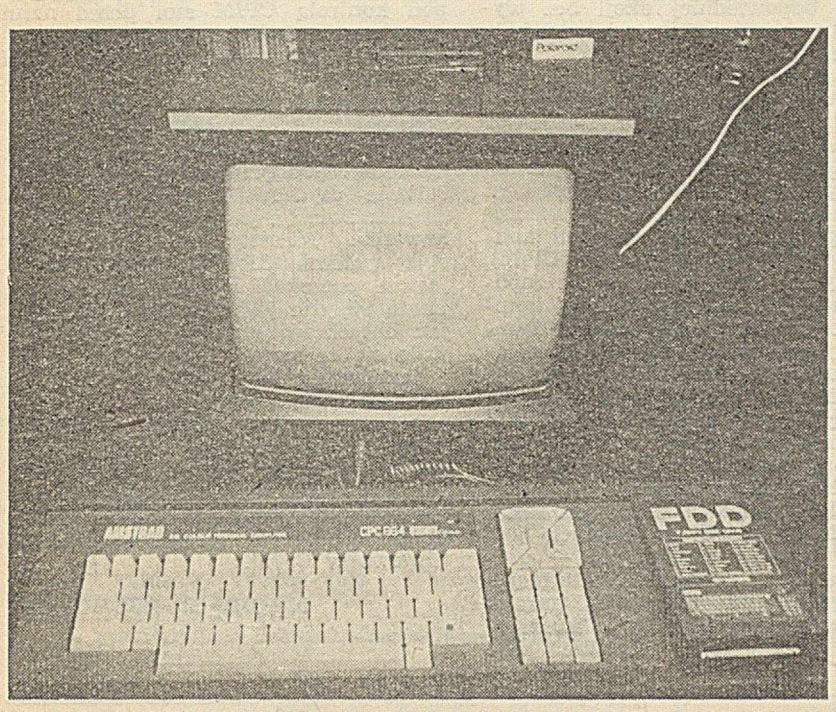
[1] Aho A. V., Hopcroft J. E., Ullmann J. D.: Projektowanie i analiza algorytmów komputerowych. PWN, Warszawa 1983

[2] Banachowski L., Kreczmar A.: Elementy analizy algorytmów. WNT, Warszawa 1983

[3] Jones C. B.: Konstruowanie oprogramowania metodą systematyczną. WNT, Warszawa 1984

[4] Levy H., Lessman F.: Równania różnicowe skończone. PWN, Warszawa 1966

[5] Mikołajczak B., Stokłosa J.: Złożoność obliczeniowa algorytmów. Wydawnictwo Politechniki Poznańskiej, 1984.



Fot. WOJCIECH JURZYK

Publikując opis pierwszego komputerowego dziecka firmy AMSTRAD, obiecaliśmy Czytelnikom bardziej szczegółowy opis CPC 464, gdy komputer ten znajdzie się „w zasięgu" redakcji. Teraz, gdy stało się to możliwe, stanęliśmy przed dylematem — na rynku brytyjskim pojawiła się nowa, nieco rozszerzona wersja tego komputera. Ponieważ firma AMSTRAD poszła w kierunku atrakcyjnym dla sporej grupy potencjalnych odbiorców w naszym kraju, zdecydowaliśmy się na opublikowanie opisu nowej wersji. Podobnie jak poprzednio, materiał został przygotowany na podstawie informacji zaczerpniętych z pism brytyjskich, z tym jednak, że autor poniższego tekstu miał okazję wcześniej „dotknąć" CPC 664, a obecnie ów komputer ma nawet na własność.

Rewelacje, o których pisał autor poprzedniego materiału o AMSTRADZIE, okazały się w pełni prawdziwe. Z mankamentów, które dostrzegliśmy w dotychczasowej eksploatacji, należy wymienić:

- **wrażliwość na niskie napięcie w sieci (w Wielkiej Brytanii jest napięcie 240 V, trudno więc mieć pretensje do komputera, że przy spadku napięcia poniżej 210 V przestaje prawidłowo funkcjonować),**
- **mała pojemność dyskietek: 170 KB zapełnia się bardzo szybko, tak więc praktycznie trzeba korzystać z dwóch napędów.**

Z niskim napięciem w krajowej sieci można sobie poradzić stosując dostępny w sklepach stabilizator napięcia dla telewizorów. Natomiast na dyski o większej pojemności trzeba będzie prawdopodobnie jeszcze trochę poczekać.

Warto dodać, że w chwili, gdy przygotowujemy do druku poniższy materiał, istnieją już dwie firmy oferujące CPC 464 na rynek krajowy, za złotówki. W różnych instytucjach pracuje już kilkadziesiąt egzemplarzy tego komputera i na razie nie odnotowano żadnej awarii!

AMSTRAD CPC 664

Bez szumnych zapowiedzi w maju 1985 r. pojawił się na rynku angielskim nowy mikrokomputer firmy AMSTRAD — CPC 664. Podobnie jak CPC 464 w pierwszych miesiącach sprzedaży — on także długo nie stoi na półkach sklepowych. Duży popyt na CPC 664 wynika z bardzo niskiej ceny, jak na system, który zawiera: mikrokomputer wraz z wbudowanym interpreterem języka BASIC, kolorowy monitor; napęd 3-calowych dysków elastycznych wraz z systemem operacyjnym CP/M i DR LOGO. Całość kosztuje poniżej 400 funtów angielskich.

CPC 664 jest konsekwentnym rozwinięciem CPC 464. Podstawowa różnica polega na zastąpieniu w CPC 464 magnetofonu przez napęd dysków elastycznych. Ze względu na konieczność doprowadzenia napięcia 12 V, zasilającego napęd dyskowy, potrzebne jest dodatkowe gniazdo, które umieszczono w nowej wersji monitorów (monochromatyczny GT 65, kolorowy CTM 644) i w modulatorze TV oznaczonym symbolem 12V DC. W gniazdo to wtyka się drugi przewód zasilający komputer. Nowe monitory i modulator mogą współpracować z poprzednią wersją komputera, ale nie na odwrót. Umieszczenie komputera i napędu dysków w jednej obudowie spowodowało konieczność przeprojektowania obudowy. Jest ona obecnie nieco wyższa niż w CPC 464. Jednocześnie zmieniono klawiaturę na bardziej estetyczną kolorystycznie. Klawisze sterujące kursorem są większe i wygodniejsze w użyciu a klawisze numeryczne, które mogą być używane także jako funkcyjne, otrzymały dodatkowe oznaczenia — cyfra została poprzedzona literą „f"; f0, f1,...,f9.

CPC 664 ma prawie identyczną architekturę jak CPC 464. Dzięki temu oprogramowanie napisane dla 464 jest przenoszalne na 664, niestety z pewnymi wyjątkami. Niektóre programy dla 464 opracowane przez niezależnych producentów oprogramowania wykorzystują wewnętrzne procedury komputera. Ponieważ ROM został trochę poprzestawiany, to wejście w procedury „na skróty", choć przyspieszało realizację programów dla CPC 464, może teraz powodować błędy przy próbie uruchomienia takich programów na 664.

CPC 664 został wyposażony w dodatkowe gniazdo magnetofonowe, które zapewnia możliwość wczytywania zapisu programów na taśmie magnetofonowej, w sposób identyczny jak dla 464. Taśmę można więc wykorzystywać do przenoszenia programów z 464, jak również do przechowywania rzadko używanych programów lub danych. CPC 664 ma również dodatkową łączówkę do przyłączenia drugiego napędu dysków elastycznych. W 464 oba napędy przyłącza się poprzez jeden kabel wychodzący z interfejsu dyskowego, wtykanego w złącze, na które wyprowadzona jest szyna systemowa. W 664 złącze to zostało zachowane, dzięki czemu można podłączyć do komputera dodatkowe interfejsy szeregowe i równoległe lub oprogramowanie zapisane na ROM-ie (dostępne już na rynku).

CPC 664 jest wyposażony w rezydencyjny interpreter LOKOMOTIVE BASIC. Jest to nieco rozszerzona wersja w stosunku do 464. Modyfikacje dotyczą komend graficznych i obsługi błędów. Dodano ważną komendę „FILL n", której bardzo brakowało użytkownikom 464. Komenda ta pozwala na wypełnienie kolorem „n" zadeklarowanego obszaru ekranu. Poza tym podano możliwość łatwego rysowania linii przerywanych i wielobarwnych (komenda MASK). Dodatkowy parametr przy komendach rysujących linie pozwala uniknąć podwójnego rysowania końców odcinków oraz pojawiania się niezamierzonych

kropek na ekranie jako pozostałości linii. Komendy GRAPHICS PEN i GRAPHICS PAPER w nowej wersji umożliwiają tworzenie efektu „przezroczystych kolorów", dostępnego dotąd tylko dla tekstu. Dodano też komendę COPYCHAR$, która wczytuje znak z ekranu i komendę CURSOR. Są one wygodne przy pisaniu programów typu procesora tekstów. Wprowadzono również kilka innych komend ułatwiających posługiwanie się ekranem monitora. Warto też wspomnieć o nowej komendzie ON BREAK CONT — ułatwiającej pisanie programów, których nie można przerwać (w 464 działanie programu można zawsze [1]) przerwać poprzez wciśnięcie klawisza ESCAPE). Nowa komenda DERR (od „disc error") pozwala na programowe rozpoznanie przyczyny wykrytego przez maszynę błędu i podjęcie odpowiedniej akcji bez wychodzenia z programu (np. detekcja braku miejsca na dysku itp.).

Wraz z CPC 664 dostarczany jest dysk systemowy, zawierający system operacyjny CP/M (wersja 2.2) i po drugiej stronie DR LOGO (Digital Research LOGO). Pewne problemy stwarza mały obszar TPA (ang. Transient Program Area — obszar przeznaczony na programy użytkowe pod kontrolą systemu operacyjnego CP/M), który wynosi 39 K. Dlatego też zapowiedziano już następną wersję komputera o pamięci RAM 128 K. Tymczasem jednak wiele firm produkujących oprogramowanie energicznie przystąpiło do adaptacji programów pracujących pod kontrolą CP/M dla obecnych wersji AMSTRADA. I tak można już dziś nabyć kompilator języków PASCAL i TURBO PASCAL oraz FORTH, zapowiedziana jest także wersja FORTRANU. Dostępnych jest wiele programów użytkowych — procesory tekstów, programy kalkulacyjne, programy wspomagające prowadzenie przedsiębiorstwa oraz kontrolę budżetu domowego. Wszystkie ww. programy dostępne są już na dyskietkach 3-calowych. Ale można też dokupić napęd dysków 5-calowych i wtedy świat programów pracujących pod kontrolą CP/M stoi przed nami otworem, niezależnie od firm przenoszących oprogramowanie na dyski 3-calowe.

Dużą zaletą CPC 664 jest dobry podręcznik. Przy opracowywaniu podręcznika dla 664 skorzystano z uwag na temat podręcznika dla 464. W rezultacie użytkownik otrzymuje podręcznik zwięzły, jasny i łatwy w użytkowaniu. Niestety na szczegółowy opis systemu CP/M i LOGO trzeba jeszcze poczekać. Firma zapowiada, że ukażą się one jeszcze latem 1986 r., a jak dotychczas punktualnie wywiązywała się ze swoich zapowiedzi.

Reasumując — AMSTRAD CPC 664 jest nie tylko połączeniem CPC 464 z napędem dysków elastycznych, ale dzięki wprowadzonym rozszerzeniom jest to praktycznie nowy komputer domowy, który dzięki atrakcyjnej cenie znajduje wielu nabywców. Jest to najtańszy obecnie system wyposażony firmowo w CP/M.

MACIEJ DORUCHOWSKI

Warszawa

[1]) W CPC 464 istnieje specjalna komenda „chronionego" zapisu programów, zabezpieczająca przed kopiowaniem przez osoby niepowołane. Po wprowadzeniu do komputera, program taki samoczynnie startuje i nie daje się przerwać. Nowa komenda została prawdopodobnie przewidziana jako zabezpieczenie przed mało wprawnym operatorem (np. programy edukacyjne pisze się również dla dzieci w wieku 3 lat, a mają one zwyczaj „próbowania" wszystkich możliwych klawiszy) — przyp. AJP.

Po dłuższej (niestety!) przerwie wznawiamy publikację materiałów o grafice komputerowej. Tym co już zdążyli zapomnieć przypominamy, że Antoni Urban dzieli się doświadczeniami zdobytymi przy realizacji implementacji standardu GKS (Graphical Kernel System) na IBM PC. Ponieważ implementacja została zrealizowana w języku C, publikowane uwagi można wykorzystać i dla innych mikrokomputerów (oczywiście wyposażonych w możliwości graficzne), jeżeli tylko dysponujemy kompilatorem języka C.

Wielolinia

Tytuł artykułu jest dosłownym tłumaczeniem angielskiego terminu **polyline**. Wielolinia jest podstawowym prymitywem reprezentacji graficznej przyjętej przez twórców normy GKS. Jest to tak charakterystyczne podejście do problemu rysowania linii, że uzasadnione stało się tworzenie nowych terminów językowych. W innych, stosowanych wcześniej językach graficznych, rozkaz rysowania linii wymagał podania współrzędnych początku i końca odcinka lub samego końca, gdy za początek przyjmowane było aktualne położenie domyślnego znacznika. W przypadku GKS wystarczy podać liczbę punktów i nazwę tablicy zawierającej współrzędne kolejnych punktów wyznaczających kontur. Kontur ten składa się z odcinków linii prostej, jest więc linią łamaną. Przed przystąpieniem do rysowania należy wyodrębnić wszystkie kontury, dla każdego zadeklarować tablicę i wypełnić współrzędnymi punktów wyznaczających kontur. Na przykład, narysowanie ramki ograniczającej obszar rysowania wymaga deklaracji tablicy o wymiarach 5×2 i wpisania do niej kolejno: (x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max}), (x_{max}, y_{min}), (x_{min}, y_{min}).

Na tablicach można wykonywać takie działania, jakie wprowadza się dla konturów, a więc: translacja, obrót, jednokładność, powinowactwo, wyznaczanie ekwidystanty, sklejanie, obcinanie i inne. Wymienione działania należy oprogramować we własnym zakresie, ponieważ norma GKS dostarcza narzędzi do animacji rysunku tylko w przypadku segmentów, o których będzie jeszcze mowa.

Wielolinia może być rysowana na dowolnych, przyłączonych do komputera urządzeniach graficznych (monitor, ploter, drukarka) z uwzględnieniem atrybutów, do których należą: kolor linii, rodzaj linii (ciągła, kreskowana, kropkowana, przerywana) oraz długość części składowej linii przerywanej. W zależności od urządzenia graficznego kolor linii oznacza: kod koloru linii oraz tła (monitor lub drukarka — kolorowe), stopień szarości (monitor lub drukarka — monochromatyczne) albo numer pióra (ploter).

Cykl artykułów dotyczących normy GKS znalazł się na łamach mikroKLANU, ponieważ jest opisem nie tyle samej normy GKS, lecz jej implementacji dla mikrokomputerów oraz problemów związanych z adaptacją jądra graficznego dla małego sprzętu informatycznego. Oprogramowanie GKS napisane zostało w języku C, ale może być traktowane jak biblioteka procedur w języku wewnętrznym i wykorzystywane w dowolnym języku programowania. Oczywiście najbardziej naturalne jest stosowanie tego samego języka, to znaczy C. Ograniczeniem narzuconym przez język programowania jest długość nazw procedur. Autor proponuje stosowanie jednej tablicy o wymiarach $n \times 2$, zamiast dwóch wektorów o długości n. Przy wymienionych odstępach od normy, rozkaz wywołania prymitywu wielolinia ma następującą składnię:

Polyline (n, p);

gdzie int n — liczba punktów, float p[ ][2] — współrzędne kolejnych punktów.

Należy podkreślić, że współrzędne podawane są w układzie rysunku wprawdzie wygodnym dla użytkownika, lecz odbiegającym od układu urządzenia graficznego.
Rozkazy deklaracji atrybutów mają postać:

Set_ltype (lt);

gdzie: char lt — rodzaj linii (lt = 0 — ciągła; lt = 1 — kreska; lt = 2 — kropka-kropka; lt = 3 — kreska-kropka)

Set_lcolour (lc);

gdzie: char lc — kod koloru linii oraz tła (lc = 0 — czarny, lc = 1 — niebieski, lc = 2 — zielony, lc = 3 — niebies-

ko-zielony, lc = 4 — czerwony, lc = 5 — fioletowy, lc = 6 — żółty, lc = 7 — biały)

Set_lsize (ls);

gdzie: float ls — długość części składowej linii przerywanej.

Atrybuty prymitywu wielolinia mogą być deklarowane sukcesywnie lub — po wcześniejszym przypisaniu zestawów atrybutów numerowi indeksu — przełączane. Zazwyczaj rysunki wykonywane są przy zastosowaniu skończonej liczby reprezentacji atrybutów. Do przygotowania i przełączania indeksu służą rozkazy:

Line_representation (li, lt, ls, lc);

gdzie: int li — numer indeksu; char lt — rodzaj linii; float ls — długość części składowej linii przerywanej; char lc — kod koloru
oraz

Set_lindex (li);

gdzie: int li — numer indeksu

Realizacja implementacji prymitywu wielolinia dla mikrokomputera wymagała rozwiązania następujących dwu problemów:

1. Jak uwzględnić rodzaj linii i długość części składowej linii przerywanej?

2. Jak rysować linię na urządzeniach graficznych bez interpolatora liniowego?

Problem 1 rozwiązano przez zakodowanie rodzaju linii w bajcie, w którym bit 1 oznacza **rysuj**, a bit 0 — **nie rysuj**. I tak poszczególnym rodzajom linii odpowiadają następujące kody:

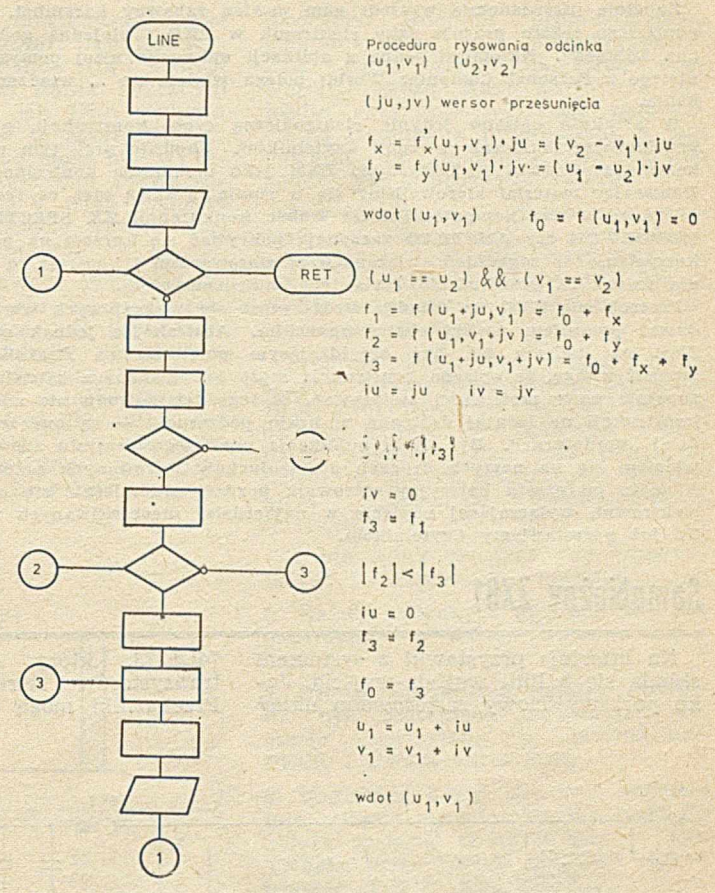
linia ciągła	--------	11111111_B
kreska	----	11110000_B
kropka-kropka	- -	10001000_B
kreska-kropka	--- -	11100100_B

Rozmiar linii uwzględniany jest przez wielokrotne (zależne od deklaracji długości części składowej linii przerywanej) testowanie tego samego bitu kodu rodzaju linii. Takie rozwiązanie problemu daje możliwość uwzględniania rodzaju linii wewnątrz procedury generacji linii.

Problem 2 rozwiązano wykorzystując algorytm przedstawiony na załączonym rysunku schematu blokowego. Algorytm przytoczony jest dla osób, które swój sprzęt mikrokomputerowy chcą wyposażyć w interpolator linii. Podana metoda jest szczególnym przypadkiem algorytmu generacji krzywych drugiego stopnia, zadanych równaniem uwikłanym:

$f(x, y) = 0$

Stosowany jest tu wzór Taylora dla funkcji dwu zmiennych rzeczywistych:



Schemat blokowy algorytmu

$$[1]\quad f(x+\Delta x, y+\Delta y) = f(x,y) + f'_x(x,y)\Delta x + f'_y(x,y)\Delta y + \frac{1}{2}(f''_{xx}(x,y)\Delta x^2 + 2f''_{xy}(x,y)\Delta x\Delta y + f''_{yy}(x,y)\Delta y^2)$$

$$[2]\quad f'_x(x+\Delta x, y+\Delta y) = f'_x(x,y) + f''_{xx}(x,y)\Delta x + f''_{xy}(x,y)\Delta y$$

$$[3]\quad f'_y(x+\Delta x, y+\Delta y) = f'_y(x,y) + f''_{xy}(x,y)\Delta x + f''_{yy}(x,y)\Delta y$$

Za pomocą wzoru [1] obliczane są wartości funkcji f dla trzech możliwych kierunków przesunięć (f_1, f_2, f_3). Spośród tych liczb wybierana jest ta, której wartość bezwzględna jest najmniejsza, co zapewnia optymalne przemieszczenie. Po dokonaniu wyboru modyfikowane są wartości pochodnych cząstkowych według wzorów [2] i [3].
Algorytm ten wykorzystany zostanie jeszcze przy omawianiu interpolatora kołowego (prymityw G.D.P.).

ANTONI URBAN
Computer Studio Kajkowscy
Gdynia

Firma IBM wypuściła nowy model komputera osobistego o nazwie IBM JX. Wszystko wskazuje, że jest to następca nieudanego modelu IBM PCjr. Dosyć zaskakujący może wydawać się fakt, że nowy komputer trafił na razie tylko na rynek... australijski. Być może jest to generalna próba przed wprowadzeniem go do Europy Zachodniej i USA. Próba o tyle sensowna, że komputer montowany jest w Japonii (gdzie oczywiście sprzedawana jest japońska wersja — mało użyteczna dla Europejczyków). Model podstawowy zawiera tylko 64 KB pamięci RAM, ale za to aż 256 KB ROM. Wynika to z faktu, że najprostsza wersja nie jest wyposażona w dyski elastyczne. Razem z monitorem kolorowym kosztuje ona 2115 dol. australijskich. Poszerzona wersja ma 128 KB RAM i napęd dysków elastycznych 3,5". Na dyskietce można zapisać 180 KB. Najbardziej rozbudowana wersja ma 256 KB RAM i dwa napędy 3,5". Kosztuje ona 3365 dol. australijskich. Konstrukcja komputera umożliwia rozszerzenie pamięci do 512 KB RAM. Możliwe jest też dołączenie dodatkowego napędu 5,25" (596 dol.), co pozwala uzyskać przenośność oprogramowania i danych z IBM PC. Z punktu widzenia firmy IBM, nowy komputer jest tani, tyle że w domowych zastosowaniach nalepka IBM nie jest taka ważna.

Zupełnie nieświadomie wyszedł nam pewien zabawny karambol. Otóż autor poniższego tekstu pracuje jako elektronik w PKP i niejedną godzinę spędził „na kółkach". Natomiast jedna z aplikacji opisanego niżej pomysłu (zaczerpniętego z Personal Computer World) polega właśnie na ... wsadzeniu ZX81 na kółka.

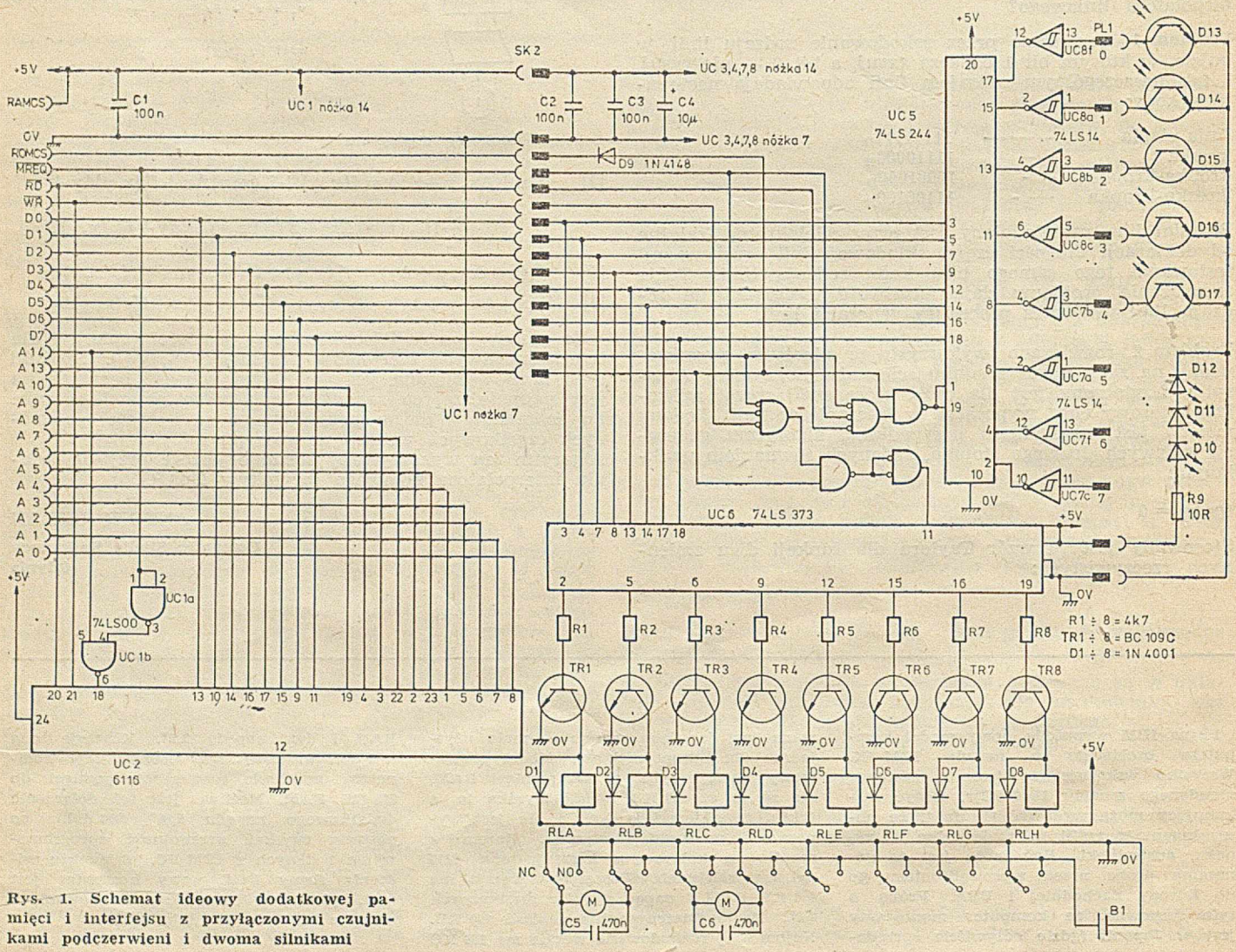
W artykule opisano jedynie elektroniczną część konstrukcji, pozostawiając problemy mechaniczne inwencji Czytelników. Chodziło przy tym raczej o pokazanie możliwości wykorzystania ZX81 jako urządzenia kontrolno-sterującego. Dobierając materiał kierowaliśmy się w sumie tą samą ideą co redakcja PCW — nawet u nas komputerki ZX81 wobec konkurencji ZX SPECTRUM, COMMODORE C64 czy AMSTRAD zaczynają pokrywać się kurzem na półkach. Wykorzystanie do zagadnień związanych ze sterowaniem i kontrolą to może ostatnia szansa na przedłużenie życia tego mikrokomputera.

Przerobienie ZX81 na dziadka stróża może się w krajowych warunkach wydawać pomysłem nazbyt ekstrawaganckim. Abstrahując jednak od tego, czy ZX81 na kółkach ma być objeżdżającym pomieszczenia strażnikiem, warto zwrócić uwagę, że opisana konstrukcja może być zasilana z baterii, co stwarza zupełnie nowe możliwości zastosowań. Z tego też powodu nie adaptowaliśmy konstrukcji na łatwiej dostępne w kraju podzespoły — są one znacznie bardziej „prądożerne". Być może publikacja poniższego pomysłu zaowocuje pojawieniem się na naszych drogach np. maluchów sterowanych przez komputer. A może po prostu bateryjny sterownik pozwoli uniezależnić się od humorów elektrowni, wyłączającej napięcie w najbardziej nieoczekiwanych momentach? To już pozostawiamy Czytelnikom.

Samobieżny ZX81

Na interfejs przystawki z systemem składa się 8 linii wejścia-wyjścia. Jako port wyjściowy zastosowano układ 74LS373, który za pośrednictwem tranzystorów steruje przekaźnikami. Przekaźniki mogą być wykorzystywane do włączania małych silników, brzęczyków itp. Zapis danych do portu można zrealizować wykonując instrukcję POKE 9000, dana. Wpisana pod adres 9000 (dziesiętnie) dana powinna być traktowana w sposób binarny, tzn. wpisanie 1 włącza przekaźnik sterowany bitem D1 (pozostałe pozostają wyłączone), wpisanie 5 włącza przekaźniki sterowane bitami D1 i D4 itp. Jako port wejściowy zastosowano układ 74LS244 umożliwiający odczyt stanu logicznego na 8 liniach.

Interfejs przystawki można umieścić w obszarze adresowym pamięci między 8 a 16 K lub 40 a 48 K. Najwygodniej jednak jest wykorzystać adres 9000 (dziesiętnie). Pod tym adresem znajduje się „cień" firmowej pamięci ROM minikomputera. Obszar z „cieniem" pamięci ROM nie jest jednak pusty. Operacje odczytu interfejsu byłyby więc zafałszowane. Aby tego uniknąć linia ROMCS jest przyłączona do linii adresowej A13 przez diodę D9. Dzięki temu pojawienie się jakiegokolwiek adresu, w którym linia A13 szyny adresowej sterowana jest poziomem wysokim, podnosi również poziom sygnału na ROMCS (do-



Rys. 1. Schemat ideowy dodatkowej pamięci i interfejsu z przyłączonymi czujnikami podczerwieni i dwoma silnikami

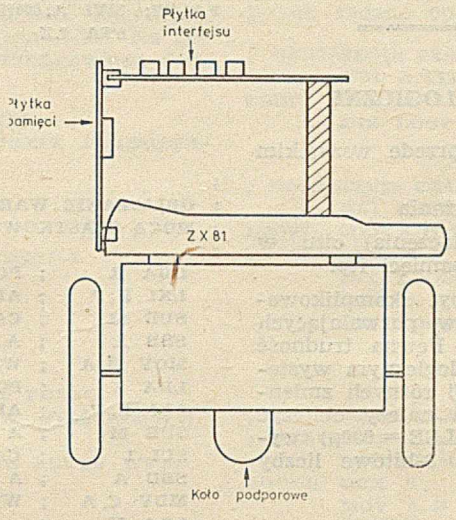
tyczy to każdego adresu z obszaru 8—16 K, zakrywanego przez ROM systemowy).

Komputer wpisuje dane do pamięci ROM sygnałami MREQ/ i WR/, aktywnymi w stanie logicznym 0. Operacja dotyczy obszaru ROM systemowego (lub jego „cienia") gdy linia A14 sterowana jest poziomem niskim. Aby przygotować układ 74LS373 (UC6) do zapisu, należy jeszcze stan logiczny 1 z wyjścia UC3b przemnożyć logicznie ze stanem 1 linii adresowej A13. Daje to gwarancję, że operacja nie dotyczy obszaru zajętego przez ROM systemowy. Z wyjścia UC4a podawany jest poziom wysoki na wejście E zatrzasku UC6. Przejście linii MREQ z powrotem na poziom wysoki wywołuje opadające zbocze na wyjściu bramki UC4a i zatrzaskuje nowe dane na liniach wyjściowych UC6. Linie te poprzez tranzystory TR1 ... 7 sterują przekaźnikami. Diody przyłączone równolegle do uzwojeń przekaźników zabezpieczają tranzystory przed przepięciami wstecznymi, występującymi przy wyłączaniu prądu w cewce.

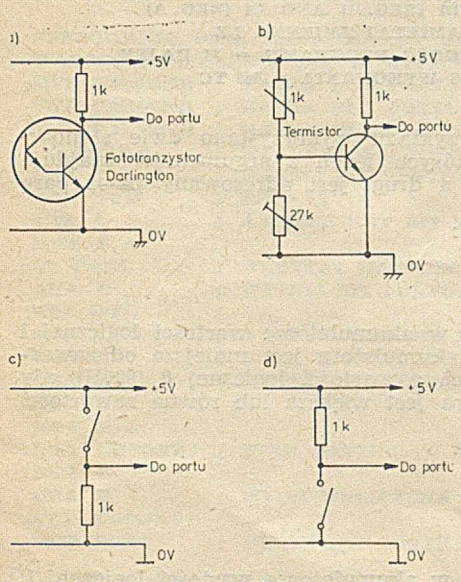
Układ wyboru portu wejściowego jest zasadniczo taki sam jak dla portu wyjściowego. Różnica polega na tym, że układ 74LS244 (UC5) pobudzany jest wtedy, gdy komputer chce czytać dane. Układ wyboru umożliwia bezpośrednie przyłączenie linii wyjściowych UC5 do szyny danych ZX81. Inwertery Schmitta (UC7 i UC8) służą do „poprawiania" zboczy sygnałów wejściowych.

Standardowa w ZX81 pamięć 1 K nie wystarcza nawet do umieszczenia w miarę prostych programów kontrolno-sterujących. Zastosowanie modułu 16 K RAM nie jest zbyt dobre ze względu na znaczny wzrost zużycia energii. Wygodnie jest zastosować tu pamięć CMOS (układ 6116 2 K × 8 CMOS RAM (UC2). W opisywanym rozwiązaniu obniża ona nawet zużycie energii przez sam ZX81. UC2 jest wybierany wtedy, gdy linie A14 i MREQ/ są równocześnie w stanie logicznym 0, tzn. identycznie jak ULA wybiera wewnętrzną pamięć w ZX81. Dlatego też linia RAMCS nie może być wykorzystywana — została ona przyłączona do napięcia +5 V (stan logiczny 1), co powoduje odcięcie wewnętrznej pamięci RAM komputera.

Do portu wejściowego można dołączyć różnego rodzaju czujniki tak, aby dostarczane przez nie informacje mogły być wprowadzane do komputera (rys. 2). Układ może, na przykład, reagować na światło (rys. 2a) lub temperaturę (rys. 2b), albo też wprowadzać sygnały z przełącznika zależnie od przyjętej dodatniej lub ujemnej logiki (rys. 2c i 2d).

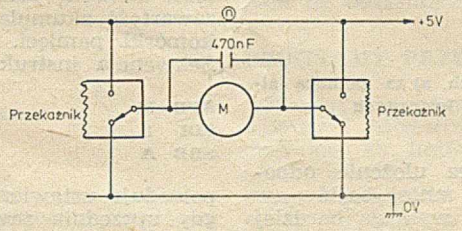


Rys. 3. Samobieżny wózek sterowany ZX81 (zasilanie z baterii 9 V), każde koło jest sterowane oddzielnym silnikiem



Rys. 2. Warianty czujników (wejść): a) czujnik świetlny, b) czujnik temperatury, c) i d) czujniki przełącznikowe

ZX81 z opisanym układem można zamontować na wózku (rys. 3). Przekaźniki wyjściowe mogą wtedy sterować silnikami napędzającymi kółka (rys. 4) i w ten sposób ZX81 może poruszać się zależnie od zewnętrznych sygnałów (temperatura, światło itp.) — stając się czymś w rodzaju robota lub sławnego niegdyś cybernetycznego żółwia. Przekaźniki wyjściowe sterują silnikami napędzającymi dwa koła (rys. 3), a sygnały wejściowe dostarczane są przez czujniki podczerwieni. Jeżeli wszystkie okna i drzwi zabezpieczymy źródłami promieniowania podczerwonego, to „zmotoryzowany" ZX81 może przyjąć na siebie obowiązki... stróża. Nasz wartownik objeżdża cyklicznie wszystkie pomieszczenia (zgodnie z programem), a gdy natrafia na „nieprawidłowe" źródło podczerwieni, uruchamia alarm.



Rys. 4. Sterowanie silnika

Jest to oczywiście jedno z zastosowań ruchomego ZX81. Zmieniając odpowiednio czujniki wejściowe, a także urządzenia, którymi steruje ZX81, można wymyśleć dla niego wiele ciekawych zajęć.

Program kontrolno-sterujący napisany w języku maszynowym (ASSEMBLER Z80) powinien zmieścić się w ok. 500 bajtach. Powiększona pamięć RAM umożliwa napisanie programu nawet w języku BASIC, pod warunkiem oszczędnego gospodarowania miejscem.

A oto kilka prostych rad zmierzających do skracania programu:

1. Nazwy zmiennych powinny być jak najkrótsze — każda litera to jeden bajt.

2. Często powtarzane numery powinny być zastąpione zmiennymi, np. często powtarzany w instrukcjach PEEK i POKE adres 9000.

3. Wszędzie, gdzie to jest możliwe (nie zmienia się odczytywana wartość) należy zastąpić IF PEEK 9000 = xxx THEN... przez LET D = PEEK 9000 i używać IF D = xxx THEN...

Przy pewnej wprawie w programowaniu takie lub podobne metody oszczędzania pamięci można wymyśleć samemu.

ROLAND HOJCZAK
Ośrodek Badań Stanu Torów PKP
Warszawa

Nowe mikrokomputery firmy AMSTRAD 6128 i 8256 mają więcej pamięci RAM, niż wynosi obszar adresowy mikroprocesora Z80. Jest to możliwe dzięki zastosowaniu pewnej sztuczki: pamięć podzielona jest na strony, a szyny mikroprocesora przełączane są pomiędzy tymi stronami. Jak można się domyślać z nazwy, model 6128 zawiera 128 KB RAM, a model 8256 ma 256 KB. Użytkowników tego ostatniego zapewne ucieszy informacja, że jego pamięć można jeszcze dwukrotnie powiększyć. Otóż po zdjęciu obudowy okazuje się, że komputer zawiera... 8 pustych podstawek. Wtajemniczeni twierdzą, iż początkowo firma AMSTRAD projektowała model 8128. W tym czasie układy pamięci DRAM o pojemności 256 Kbitów tak staniały (ok. 1,5 dol. za sztukę), że bardziej opłacalne było zamontowanie 8 układów 256 Kb niż 16 układów 64 Kb! W ten sposób zostało wolne miejsce, które po wykonaniu kilku połączeń może zostać wykorzystane dla rozszerzenia pamięci. Dodatkowa pamięć może zostać wykorzystana na tzw. RAM-disc.

Skromne łamy mikroKLANU powstrzymują nas od wykładania podstaw języka ASSEMBLER. Nie oznacza to jednak, że pozostawiamy początkujących zupełnie bez opieki. Liczymy, że Czytelnicy potrafią znaleźć w jakiejś książce znaczenie poszczególnych rozkazów ASSEMBLERA 8080. Poniższy dwuczęściowy materiał pokazuje natomiast jak programować. Prezentowane procedury nie były optymalizowane ani pod kątem czasu realizacji, ani miejsca zajmowanego w pamięci — pozostawiamy to już jako ćwiczenie wprawiające. Chodziło nam o klarowne ukazanie zapisu algorytmów realizujących pewne użyteczne funkcje. O ile procedury realizujące funkcje logiczne przedstawiają przykładowe rozwiązania pewnych szczególnych problemów, o tyle algorytmy opóźnień czasowych i konwersji kodów mogą zostać włączone do zestawu procedur bibliotecznych, podobnie jak opisywane w poprzednim odcinku procedury arytmetyczne.

Przykłady procedur w języku ASSEMBLER 8080 (2)

PROGRAMY REALIZUJĄCE FUNKCJE LOGICZNE

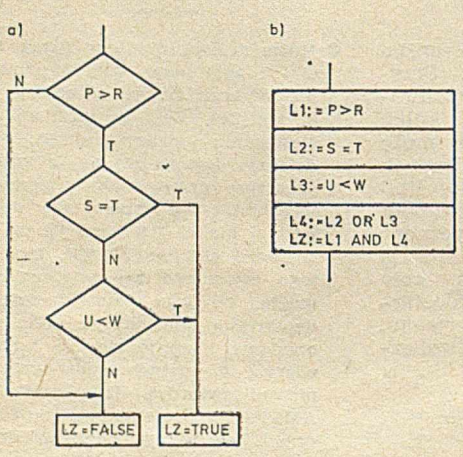
Funkcje logiczne wykorzystywane są przede wszystkim w dwóch przypadkach:

- dla wyliczenia wartości logicznej wyrażenia
- dla wysterowania (włączenia lub wyłączenia) bitu w porcie zewnętrznym, rejestrze, komórce pamięci itp.

Pierwszy przypadek nie wydaje się zbyt skomplikowany. Mikroprocesor 8080 ma szereg rozkazów, pozwalających na realizację każdej operacji logicznej. Pewna trudność pojawia się wówczas, gdy w wyrażeniu logicznym występuje wiele operatorów logicznych i relacji różnych zmiennych. Rozpatrzmy przypadek, w którym należy obliczyć wartość logiczną (TRUE = $0FF_H$ lub FALSE = 000_H) wyrażenia logicznego, w którym występują 8-bitowe liczby całkowite bez znaku.

LZ = P > R AND (S = T OR U < W)

Wartość zmiennej logicznej Z można obliczyć korzystając z jednego ze sposobów przedstawionych na rysunku 1.



Rys. 1. Sposoby obliczania funkcji logicznych a) za pomocą algorytmu; b) za pomocą cząstkowych operacji logicznych

Obliczanie wartości wyrażenia poprzez ułożenie odpowiedniego algorytmu zajmuje z reguły mniej bajtów w pamięci i wykonuje się szybciej, jednak wymaga bardziej wnikliwej analizy wyrażenia i dlatego częściej stosowane jest przez programistów, piszących program w języku symbolicznym, niż przez kompilatory języków wyższego poziomu, zamieniające postać wyrażenia na kod maszynowy. Przykłady programów realizujących obie metody obliczania wyrażeń logicznych przedstawiono poniżej.

```
; OBLICZANIE WARTOŚCI WYRAŻENIA LOGICZNEGO ZA POMOCĄ ALGORYTMU
;
        LDA R           ; POBIERZ ZMIENNA R
        LXI H,P         ; ADRES ZMIENNEJ P
        CMP M           ; POROWNAJ P I R
        JNC FALSE
        LDA S           ; POBIERZ ZMIENNA S
        LXI H,T         ; ADRES ZMIENNEJ T
        CMP M           ; POROWNAJ S I T
        JZ TRUE
        LDA U           ; POBIERZ ZMIENNA U
        LXI H,W         ; ADRES ZMIENNEJ W
        CMP M
        JNC FALSE
TRUE:   MVI A,0FFH      ; WARTOŚĆ TRUE
        STA LZ          ; ZMIENNA LZ ROWNA TRUE
        ...
FALSE:  MVI A,000H      ; WARTOŚĆ LOGICZNA FALSE
        STA LZ          ; ZMIENNA LZ ROWNA FALSE
                        ; DŁUGOŚĆ PROGRAMU = 40 BAJTOW
                        ; CZAS WYKONANIA = 57 TC do 140 TC
;
; OBLICZANIE WARTOŚCI WYRAŻENIA LOGICZNEGO ZA POMOCĄ CZĄSTKOWYCH OPERACJI LOGICZNYCH
;
    LDA R       ; POBIERZ ZMIENNA R
    LXI H,P     ; ADRES ZMIENNEJ P
    SUB M       ; CARRY = 1 GDY P WIEKSZE OD R
    SBB A       ; A = 0FFH GDY P WIEKSZE OD R
    MOV B,A     ; WYNIK CZASTKOWY L1
    LDA S       ; POBIERZ ZMIENNA S
    LXI H,T     ; ADRES ZMIENNEJ T
    SUB M       ; A = 000H GDY S ROWNE T
    SUI 1       ; CARRY = 1 GDY S ROWNE T
    SBB A       ; A = 0FFB GDY S ROWNE T
    MDV C,A     ; WYNIK CZASTKOWY L2
    LDA U       ;
    LXI H,W     ; ADRES ZMIENNEJ W
    SUB M       ; CARRY = 1 GDY U MNIEJSZE OD W
    SBB A       ; A = 0FFH GDY U MNIEJSZE OD W
    ORA C       ; L4 (REG. A) = L2 (REG. C) OR L3 (REG. A)
    ANA B       ; A = L1 (REG. B) AND L4 (REG. A)
    STA LZ      ; ZAPAMIETAJ ZMIENNA LZ
                ; DŁUGOŚĆ PROGRAMU = 33 BAJTY
                ; CZAS WYKONANIA = 140 TC
```

W drugiej części przykładu wykorzystano dwie standardowe operacje, w których jeden z argumentów znajduje się w akumulatorze, a drugi jest adresowany przez parę rejestrów HL.
Sekwencja instrukcji:

```
SUB M
SBB A
```

powoduje ustawienia w akumulatorze wartości logicznej 1 (0FFH) gdy wartość akumulatora jest mniejsza od zawartości komórki pamięci, a wartości logicznej 0 (000H) gdy zawartość akumulatora jest większa lub równa zawartości komórki pamięci.
Sekwencja instrukcji:

```
SUB M
SUI 1
SBB A
```

powoduje ustawienie w akumulatorze wartości logicznej 1, gdy uprzednia zawartość akumulatora jest równa zawartości komórki pamięci M (o adresie zawartym w rejes-

trach H i L), a w przeciwnym przypadku ustawienie w akumulatorze wartości logicznej 0.

Przypadek wykorzystania funkcji logicznych dla ustalenia lub kasowania wybranego bitu stwarza więcej kłopotów z uwagi na brak w procesorze 8080 rozkazów operujących na bitach. Z braku innych możliwości muszą być tu wykorzystane w odpowiedni sposób rozkazy operujące na całych bajtach. Dla „zapalenia" („zgaszenia") wybranego bitu w odpowiednim rejestrze, porcie lub komórce pamięci może być wykorzystany rozkaz sumy logicznej (iloczynu logicznego) z maską, w której jest „zapalony" („zgaszony") tylko wybrany bit. Ilustruje to poniższy przykład:

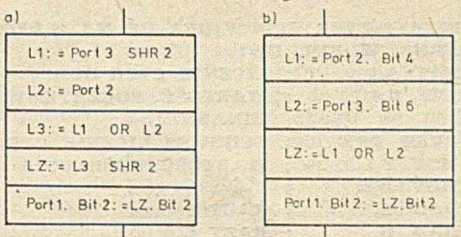
```
; ROZKAZY ZAPALAJĄCE I GASZĄCE CZWARTY (A3) BIT
  AKUMULATORA
;
  ORI 00001000B   ; ZAPAL BIT 4 AKUMULATORA
  .....
  ANI 11110111B   ; ZGAŚ BIT 4 AKUMULATORA
```

Często w programach realizujących sterowanie sekwencyjne stan wybranej linii (bitu) portu wejściowego jest funkcją stanów linii (bitów) portów wejściowych. W tym wypadku przed wysterowaniem odpowiedniej linii należy wyliczyć wartość logiczną, jaką należy jej przypisać. Podobnie jak w przypadku obliczania wartości logicznych wyrażeń, można to zrealizować na kilka sposobów. Na rysunku 2 oraz w poniższym przykładzie przedstawiono dwa sposoby rozwiązania problemu:

```
  PORT1—BIT2 = PORT2—BIT4 OR PORT3—BIT6
; STEROWANIE BITU WYJŚCIOWEGO POPRZEZ PRZESUWA-
; NIE BITÓW WEJŚCIOWYCH
  MVI B,00001000B  ; MASKA DLA PORTU 2
  MVI C,00100000B  ; MASKA DLA PORTU 3
  MVI D,11111101B  ; MASKA DLA PORTU 1
  IN PORT 2        ; WCZYTAJ WARTOŚĆ PORTU 2
  ANA B            ; WYTNIJ BIT PORTU 2
  MOV B, A
  IN PORT 3        ; WCZYTAJ PORT 3
  ANA C            ; WYTNIJ BIT PORTU 3
  RRC
  RRC              ; PRZESUŃ NA POZYCJE PORTU 2
  ORA B            ; SUMA BITÓW PORTÓW WEJŚCIOWYCH
  RRC
  RRC              ; PRZESUN NA POZYCJE PORTU WYJŚ-
                     CIOWEGO
  MOV B, A
  LDA ZMPORT 1     ; STAN PORTU 1
  ANA D            ; WYKASUJ BIT PORTU 1
  ORA B            ; WPISZ WYLICZONA WARTOŚĆ
  STA ZMPORT 1
  OUT PORT 1       ; USTAW BIT PORTU 1

; STEROWANIE BITU WYJŚCIOWEGO POPRZEZ WYLICZENIE
; WARTOŚCI LOGICZNEJ BITÓW WEJŚCIOWYCH
  MVI B,00001000B  ; MASKA DLA PORTU 2
  MVI C,00100000B  ; MASKA DLA PORTU 3
  MVI D,11111101B  ; MASKA DLA PORTU 1
  IN PORT 2        ; WCZYTAJ WARTOŚĆ PORTU 2
  ANA B            ; WYTNIJ BIT PORTU 2
  ADI 0FFH         ; BIT = 0 ?
  SBB A            ; A = 0FFH GDY BIT ROWNY 1
  MOV B, A
  IN PORT 3        ; WCZYTAJ WARTOŚĆ PORTU 3
  ANA C            ; WYTNIJ BIT PORTU 3
  ADI 0FFH         ;
  SBB A            ; A = 0FFH GDY BIT ROWNY 1
  ORA B            ; SUMA Z WARTOSCIA BIT PORTU 2
  ORA D            ; WYTNIJ BIT PORTU 1
  XRA D
  MOV B, A
  LDA ZMPORT 1     ; STAN PORTU 1
  ANA D
  ORA B            ; WPISZ WYLICZONA WARTOSC
  STA ZMPORT 1
  OUT PORT 1       ; USTAW BIT PORTU 1
```

Drugi przedstawiony w przykładzie sposób, mimo że dłuższy, jest bardziej uniwersalny, gdyż kod procedury nie zależy od pozycji bitów w portach wejściowych i wyjściowych.



Rys. 2. Sposób sterowania linii portu wyjściowego w funkcji stanu linii wejściowych a) przez przesuwanie bitów wejściowych; b) przez wyliczanie wartości logicznej

PROGRAMY REALIZACJI FUNKCJI CZASOWYCH

W bardziej złożonych systemach do pomiaru czasu jest wykorzystywany specjalizowany układ (programowany zespół liczników). W prostszych zastosowaniach, gdy procesor nie jest zaangażowany w realizację dodatkowych zadań, może on zająć się zliczaniem cykli zegarowych. Znając czas wykonywania poszczególnych instrukcji można napisać procedurę (trwającą przez wymagany czas). Poniżej przedstawiono przykłady programów (pętli czasowych) realizujących żądane opóźnienia.

```
; OPÓŹNIENIE CZASOWE Z 1 REJESTREM
        MVI A, TIM  ; PARAMETR PĘTLI
LOOP:   DCR A
        JNZ LOOP    ; DŁUGOŚĆ = 7 BAJTÓW
                    ; CZAS = 7 + 15 + TIM (10 µs do 1847 µs)

; OPÓŹNIENIE CZASOWE Z 2 REJESTRAMI
        MVI C,TIM1  ; ZEWNĘTRZNY PARAMETR PĘTLI
LOOP1:  MVI A,TIM2  ; WEWNĘTRZNY PARAMETR PĘTLI
LOOP2:  DCR A
        JNZ LOOP2   ; PĘTLA WEWNĘTRZNA
        DCR C       ; PĘTLA ZEWNĘTRZNA
        JNZ LOOP1   ; DŁUGOŚĆ = 12 BAJTÓW
                    ; CZAS = 7 + TIM1* (22 + 15*TIM2)
                    ; 44TC DO 988679 TC (21 µs DO 474 ms)

; OPÓŹNIENIE CZASOWE Z 3 REJESTRAMI
; PRZYKŁAD 1
        LXI B,TIM3  ; PARAMETR PĘTLI
LOOP3:  DCX B       ;
        MOV A, B
        ORA C       ; DŁUGOŚĆ = 9 BAJTÓW
        JNZ LOOP3   ; CZAS = 10 + 24*TIM3
                    ; 34 TC DO 157287 TC (16 µs DO 775 ms)
; PRZYKŁAD 2
        MVI B,TIM4  ; PARAMETR PĘTLI ZEWNĘTRZNEJ
LOOP4:  MVI C,TIM5  ; PARAMETR PĘTLI ŚRODKOWEJ
LOOP5:  MVI A,TIM6  ; PARAMETR PĘTLI WEWNĘTRZNEJ
LOOP6:  DCR A
        JNZ LOOP6   ; PĘTLA WEWNĘTRZNA
        DCR C
        JNZ LOOP5   ; PĘTLA ŚRODKOWA
        DCR B
        JNZ LOOP4   ; PĘTLA ZEWNĘTRZNA
                    ; DŁUGOŚĆ = 18 BAJTÓW
                    ; CZAS = 7 + TIM4* (22 + TIM5* (22 +
                      + 15*TIM6))
                    ; 66 TC DO 253 105664 TC (32 µs do 121 sek)
```

Wykorzystując programową metodę realizacji opóźnienia czasowego należy pamiętać o dodatkowych opóźnieniach wnoszonych przez pozostałe ścieżki programu, jeżeli zależy nam na dokładnym odmierzeniu czasu.

PROGRAMY PRZETWARZANIA KODÓW

Poniżej przedstawiono programy zmiany kodów BCD na binarny i odwrotnie. Z uwagi na prostsze operacje arytmetyczne na liczbach binarnych, a jednocześnie bardziej komunikatywne — dla operatora — posługiwanie się liczbami dziesiętnymi, operacja zamiany z kodu BCD na binarny odbywa się po wprowadzeniu danych, a przejście z kodu binarnego na BCD przed wyprowadzeniem danych.

```
; KONWERSJA 4 CYFR BCD W REJ. BC NA 16-BITOWĄ LICZ-
; BĘ BINARNĄ W REJ. HL
BCDHEX: MVI A,4         ; LICZNIK CYFR BCD
        LXI H,0         ; WARTOŚĆ POCZĄTKOWA LICZBY
                          BINARNEJ
BCDH1:  PUSH PSW        ; SCHOWAJ LICZNIK CYFR
        MOV E,L         ; WARTOŚĆ POŚREDNIA WYNIKU
        MOV D,H         ;
        DAD H           ; RAZY 2
        DAD H           ; RAZY 4
        DAD H           ; RAZY 5
        DAD H           ; RAZY 10
        LXI D,400H      ; LICZNIK BITÓW KOLEJNEJ CYFRY
BCDH2:  MOV A,C
        RAL             ; PRZESUŃ LICZBĘ BCD O 1 BIT
        MOV C,A
        MOV A,B
        RAL
        MOV B,A
        MOV A,E
        RAL             ; WPROWADŹ BIT DO REJ. E
        MOV E,A
        DCR D           ; WSZYSTKIE 4 BITY
        JNZ BCDH2
        DAD D           ; DODAJ BCD DO HEX
        POP PSW         ; LICZNIK CYFR BCD
        DCR A
        JNZ BCDH1
        RET

                        ; DŁUGOŚĆ = 35 BAJTÓW
                        ; CZAS WYKONANIA = 0,75 ms

; KONWERSJA 16-BITOWEJ LICZBY BINARNEJ Z REJ. HL
; NA 4 CYFRY BCD W REJ. DE
BINBCD: XRA A           ; WARTOŚĆ POCZĄTKOWA
        LXI B,0FC18H    ; —1000
        CALL ILORAZ
        CPI 0AH         ; LICZBA TYSIĘCY WIĘKSZA
        JNC ERROR       ; OD 10 ?
        LXI B,0FF9CH    ; —100
        CALL ILORAZ
        MOV D,A         ; LICZBA TYSIĘCY I SETEK
        MVI C,0F6H      ; —10
        CALL ILORAZ
        MVI C,0FFH      ; —1
        CALL ILORAZ
        MOV E,A         ; LICZBA JEDNOSTEK I DZIESIĄTEK
        RET
;
ILORAZ: ADD A           ; BIT W LEWO
        ADD A           ; 2 BITY W LEWO
        ADD A           ; 3 BITY W LEWO
        ADD A           ; 4 BITY W LEWO
ILOR1:  PUSH H          ; ZAPAMIĘTAJ RESZTĘ
        DAD B           ; ODEJMIJ JEDNOSTKĘ
        JNC ILOR2
        XTHL            ; MIEŚCI SIĘ
        POP H           ; RÓŻNICA
        INR A           ; WYNIK CZĘŚCIOWY
        JMP ILOR1
ILOR2:  POP H           ; POBIERZ RÓŻNICĘ
        RET

        ; DŁUGOŚĆ = 48 BAJTÓW
        ; CZAS WYKONYWANIA OD 0,5 ms
          DO 2,5 ms
```

Program konwersji liczby z postaci binarnej na BCD sprawdza, czy zamieniana liczba mieści się w czterech pozycjach kodu BCD.

ADAM PLUTA
Warszawa

Czytelnicy, którzy zapoznali się z możliwościami mikroprocesora 6809, zapewne zadają sobie pytanie: czy dostępne jest dla niego gotowe oprogramowanie? Z oprogramowaniem układów firmy MOTOROLA nie jest u nas w kraju najlepiej. Nie wynika to jednak z faktu, że oprogramowania tego w ogóle nie ma. Stosunkowo niewielka popularność mikroprocesorów z serii 6800 (trudniejsze do zdobycia niż np. Z80) ogranicza obieg tego oprogramowania do wąskiego kręgu zainteresowanych.

W poniższym tekście przedstawiamy dość interesujący pakiet programowy, który upraszcza realizację stosunkowo szerokiej klasy zagadnień. Faktycznie, dysponując tego typu zapleczem, jak układ 6839, jest znacznie łatwiej podjąć decyzję o zastosowaniu w planowanej konstrukcji mikroprocesora 6809.

Norma IEEE „w pigułce"

Od pewnego czasu firmy produkujące układy scalone starają się wyjść naprzeciw potrzebom użytkowników poszukujących taniego, standardowego oprogramowania. Stwierdzono, że przy stale rosnących kosztach oprogramowania jedyną drogą obniżenia cen jest masowość produkcji. Recepta bez wątpienia prosta: należy sprzedawać standardowe oprogramowanie w postaci pamięci stałej, programowanej u producenta. Przy masowej produkcji koszty takiego oprogramowania będą bliskie kosztom wytwarzania układu scalonego. Tak postąpiono z systemem operacyjnym CP/M integrując go w jednym układzie z mikroprocesorem Z80. Ostatnio firma MOTOROLA wypuściła na rynek układ MC6839 — pamięć stałą o pojemności 8 K × 8, w której zawarta jest pełna implementacja normy IEEE dotyczącej operacji zmiennoprzecinkowej. Oprogramowanie to przeznaczone jest dla procesora MC6809.

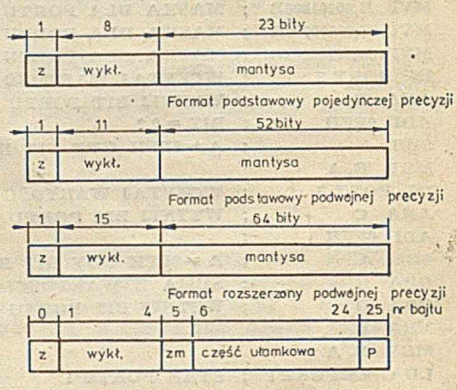
Dość szczegółowy opis tej normy znajdzie Czytelnik w numerze 1'1985 INFORMATYKI. My zatrzymamy się na sposobie, w jaki wskazania normy zostały spełnione w układzie 6839 oraz na przyjętej metodzie komunikacji programu użytkownika z pakietem zmiennoprzecinkowym.

Układ 6839 ma stanowić alternatywę dla rozwiązania sprzętowego w postaci koprocesora lub układu specjalizowanego, np. Am 9511. Oczywiście rozwiązanie oferowane przez 6839 jest o kilka rzędów wolniejsze, ale — również proporcjonalnie tańsze. Projektanci, pragnąc dodatkowo zrekompensować użytkownikowi długi czas obliczeń, wprowadzili — w porównaniu do normy — pewne rozszerzenia. Dotyczą one przede wszystkim diagnostyki błędów i liczby realizowanych operacji.

Program zawarty w układzie napisany jest w sposób przemieszczalny, tzn. działa niezależnie od położenia układu przestrzeni adresowej pamięci procesora. Również „notatnik" (obszar roboczy) w pamięci RAM może być dowolnie umieszczony. Program wykorzystuje 185 komórek na stosie systemowym.

Zgodnie z normą, zmiennoprzecinkowe argumenty operacji występować mogą w następujących formatach (rys. 1):

- podstawowym pojedynczej precyzji
- podstawowym podwójnej precyzji
- rozszerzonym podwójnej precyzji
- rozpakowanym BCD.



Rys. 1. Formaty zmiennoprzecinkowe
z — znak wykładnika, zm — znak części ułamkowej, p — pozycja kropki dziesiętnej

Ponadto występują formaty stałoprzecinkowe pojedynczej i podwójnej precyzji. Stanowią one, obok formatu

BCD, rozszerzenie w stosunku do ustaleń normy.

Wybór sposobu zaokrąglania wyników zależy od parametrów przekazywanych do pakietu. Przewidziano następujące sposoby zaokrąglania:

- do liczby najbliższej
- w kierunku zera (jest to zgodne z wymogami implementacji FORTRANU)
- w kierunku $+\infty$
- w kierunku $-\infty$

Spełniony jest postulat mówiący o tym, aby w czasie obliczeń zaokrąglanie wystąpiło tylko raz. Nieskończoność może występować w postaci afinicznej bądź rzutowej. Poza tym istnieje specjalny wzorzec dla nieliczb (tzn. takich ciągów bitów, które nie mogą zostać potraktowane jako liczba w żadnym z formatów). Istnieją też określone sposoby przedstawiania liczb nieznormalizowanych i zera.

Procedury rozpoznaja siedem sytuacji wyjątkowych. Każda z nich może spowodować przerwanie programowe bądź przyjąć założony z góry wynik i kontynuować obliczenia. W przypadku przerwania sterowanie przekazywane jest do programu użytkownika, pod uprzednio podany adres.

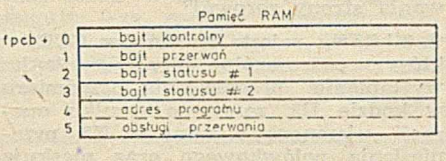
Program zawarty w 6839 rozpoznaje następujące sytuacje wyjątkowe: nielegalna operacja, niedomiar, nadmiar, dzielenie przez zero, niedokładny wynik, nadmiar stałoprzecinkowy, porównywanie liczby z nieliczbą. Przy przerwaniu pakiet dostarcza użytkownikowi informacji o przyczynie przerwania, adresie w programie użytkownika, z którego nastąpił skok do pakietu, argumentach operacji, kodzie operacji zmiennoprzecinkowej oraz proponowanym wyniku zastępczym. Dane te przekazywane są na stosie.

Operacje wykonywane przez program w 6839 mogą być jedno- lub dwuargumentowe. Są to:

- dodawanie, odejmowanie, mnożenie i dzielenie
- wyznaczanie reszty z dzielenia
- różne typy porównań
- pierwiastek kwadratowy
- wyznaczanie części całkowitej liczby
- konwersje formatów
- moduł liczby
- zamiana znaku
- przesłanie wraz z konwersją formatu.

Operacje porównania nie wymagają, aby liczby miały ten sam format.

Istnieją dwa sposoby komunikowania się programu użytkownika z pakietem zmiennoprzecinkowym. Pierwszy — rejestrowy — polega na przekazywaniu adresów argumentów w rejestrach. Drugi — stosowy — przekazuje argumenty na stosie. Obok argumentów do programu zawartego w 6839 należy przesłać adres obszaru roboczego w pamięci RAM, tzw. fpcb. Jego postać przedstawiona jest na rysunku 2.



Rys. 2. Notatnik w pamięci RAM

W bajcie kontrolnym muszą znaleźć się dane dotyczące formatu argumentów, sposobu traktowania ∞, sposobu zaokrąglania i normalizowania. Bajt przerwań stanowi maskę dla wywoływania przerwań przez określone sytuacje. Dwa bajty statusu zapisywane są przez program 6839 i stanowią opis toku operacji. W przypadku przerwania sterowanie przekazywane jest pod adres zawarty w komórkach fpcb+4 i fpcb+5. Wejścia do pakietu znajdują się w następujących miejscach: adres początkowy MC6839 + $3D_H$ dla komunikacji rejestrowej i dwie komórki dalej — dla stosowej. Za rozkazem skoku do podprogramu zmiennoprzecinkowego powinien znaleźć się bajt będący kodem operacji. Dla przykładu, wywołania procedury mnożenia mogą wyglądać jak następuje:

1) rejestr U := adres arg. 1
   rejestr Y := adres arg. 2
   rejestr D := adres fpcb
   rejestr X := adres wyniku
   LBSR rom + 3D ; „daleki" skok do podprogramu
   ⟨kod operacji mnożenia⟩
2) argument 1 na stos
   argument 2 na stos
   adres fpcb na stos
   LBSR rom + 3F
   ⟨kod operacji mnożenia⟩
   wynik ze stosu.

Jak widać, uruchomienie operacji zmiennoprzecinkowej nie jest trudne. A jak długo trwają operacje? Zależy to od częstotliwości zegara systemowego i od wielkości argumentów. Przykładowo, dla zegara 2 MHz mnożenie trwa maksymalnie 4,8 *us* (format rozszerzony), a konwersja z tego formatu na format BCD (czyli przygotowanie do wyprowadzenia w postaci ciągu kodów ASCII) — maksymalnie 85 *us*. W większości przypadków są to wielkości do przyjęcia.

Kończąc — pozostaje życzyć sobie i wszystkim Czytelnikom mikroKLANU, aby podobne programy, nawet niekoniecznie w postaci pamięci ROM 8 K × 8, produkowanej przez CEMI (sic!), pojawiły się dla procesorów, na których opiera się rodzimy konstruktor. Autor jest jednak pod tym względem sceptyczny.

JERZY ORKISZEWSKI
Warszawa

Po tegorocznych Targach Poznańskich oczy konstruktorów sprzętu zwrócone są na mikroprocesor 16-bitowy 8086. Okazało się bowiem, że firmy polonijne potrafiły w przeciągu zaledwie kilku miesięcy przekopiować zachodnie rozwiązania, mimo ich stosunkowo dużego stopnia zaawansowania technicznego.

O wadach i zaletach mikroprocesora 8086 już niebawem będzie można przeczytać na łamach mikroKLANU. Zanim to jednak nastąpi, chcemy zwrócić uwagę Czytelników na inny mikroprocesor. Chodzi o to, by nie przechodzić bezkrytycznie na technikę 16-bitową tam, gdzie nie jest to w pełni uzasadnione. Konstrukcje 16-bitowe są znacznie droższe i to nie tylko ze względu na cenę samego mikroprocesora. Tymczasem w wielu zastosowaniach zwykła „ósemka" z powodzeniem da sobie radę. Trzeba tylko pamiętać, że świat mikroprocesorów 8-bitowych nie kończy się na 8080.

Mikroprocesor 6809 cechuje stosunkowo duża moc przetwarzania, może więc być użyteczny np. w sterowniku obciążonym zagadnieniami optymalizacji lub złożonych algorytmach kompensacyjnych itp.

Nie polecamy natomiast 6809 do budowy komputerów osobistych. Nie dlatego, że nie nadaje się on do tego typu zastosowań. Po prostu dla komputerów pracujących pod kontrolą systemu operacyjnego CP/M lub kompatybilnych z IBM PC istnieje już znaczna liczba programów użytkowych, których nie opłaca się pisać jeszcze raz.

MC6809 — w pół kroku między 6800 a 68000

Procesor 6809 wart jest przedstawienia z kilku powodów. Po pierwsze, stanowi wersję rozwojową popularnego 6800, prekursora rodziny. Po drugie, pomimo 8-bitowej szyny danych ma rozbudowaną grupę rozkazów operujących na danych 16-bitowych, co czyni zeń procesor „quasi 16-bitowy". Po trzecie, i to jest właśnie jego cechą najważniejsza, ma bardzo rozbudowane tryby adresowania oraz rozkazy i mechanizmy spotykane właściwie dopiero w mikroprocesorach 16-bitowych. Dostępne rozkazy pozwalają na pisanie programów zwięzłych i nowoczesnych; w szczególności przemieszczalnych (ang. position-independent), współużywalnych i o modularnej budowie. Znacznie ułatwione jest tworzenie programów działających w czasie rzeczywistym oraz oprogramowania systemowego, w tym kompilatorów języków wysokiego poziomu.

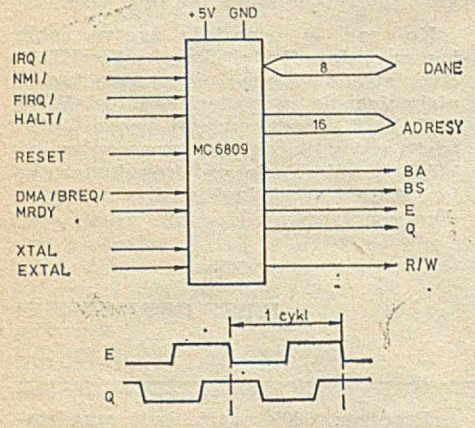
Procesor kosztuje obecnie ok. 18 marek zachodnioniemieckich, a więc

jest kilkakrotnie tańszy niż 8088 czy 68008.

Mikroprocesor MC 6809 pojawił się w 1978 r. Pierwotnie jego perspektywy rynkowe, wobec istniejących już wtedy mikroprocesorów 16-bitowych, oceniane były dość sceptycznie. Rzeczywistość okazała się jednak wcale nie taka czarna. Obecnie jest on produkowany przez kilka firm (z europejskich — EFCIS), a jego sprzedaż osiąga 3 mln sztuk rocznie.

Szyny i sygnały sterujące

Wyprowadzenia procesora przedstawione są schematycznie na rysunku. Procesor ma 16-bitową szynę adresową i 8-bitową szynę danych. Wbudowany generator wymaga jedynie uzupełnienia kwarcem o częstotliwości 4-krotnie większej niż częstotliwość szyny. O kierunku przesłania danych decyduje linia R/—W. Linie E i Q to sygnały zegarowe. W stosunku do 6800 nowością jest sygnał Q (rys.). Układ ma trzy wejścia przerwań sprzętowych. Obok typowych linii IRQ/ i NMI/ występuje linia FIRQ/. Wejście to ma wyższy priorytet niż IRQ/, a przyjęcie tego przerwania poprzedza przesłanie na stos wyłącznie licznika rozkazów i rejestru warunków. Przyspiesza to znacznie obsługę przerwania.



Wyprowadzenia MC6809 i przebiegi zegarowe

Stan procesora wskazywany jest przez poziomy na liniach BA i BS. Różnicowane są następujące stany: praca normalna, cykl przyjęcia przerwania, oczekiwanie na zewnętrzny sygnał synchronizujący oraz potwierdzenia zwolnienia szyn.

Szyny udostępniane są w wyniku rozpoznania niskiego poziomu na linii DMA/-BUS REQ/. Linia MRDY służy do współpracy z wolnymi pamięciami. Procesor zerowany jest sygnałem RESET/.

W porównaniu do 6800 wyeliminowano linię Valid Memory Address. Stało się to możliwe, ponieważ wszystkie cykle szyny 6809 są „ważne". Jeżeli procesor nie odwołuje się w danym cyklu do pamięci, to ustawia linie A0—A15 oraz R/—W w stan wysoki, nie dokonując jednak odczytu. Sygnał Q może służyć do synchronizacji w systemie, nie jest jednak niezbędny do sterowania układów peryferyjnych rodziny M6800.

Rejestry procesora

Programista ma dostęp do ośmiu rejestrów. Są to:

- dwa 8-bitowe akumulatory oznaczane jako A i B. Przy wykonywaniu rozkazów 16-bitowych oba te rejestry tworzą jeden 16-bitowy akumulator D
- 16-bitowy licznik rozkazów
- cztery 16-bitowe rejestry indeksowe X, Y, S i U. Dwa z tych rejestrów są również wskaźnikami stosów: systemowego rejestru (S) i użytkownika rejestru (U)
- 8-bitowy rejestr strony zerowej DP. Jego zawartość tworzy bardziej znaczący bajt adresu w trybie „adresowania strony zerowej"
- 8-bitowy rejestr warunków CCR, w którym poszczególne bity sygnalizują: wystąpienie przeniesienia, nadmiaru w kodzie U2, zerowego wyniku operacji, ujemnego wyniku w U2, przeniesienia połówkowego oraz maskują przerwania IRQ/ i FIRQ/. Ostatni bit wskazuje, czy ostatnio przyjęte przerwanie było zwykłe czy „szybkie" (FIRQ/).

Na podkreślenie zasługuje istnienie dwu odrębnych stosów, co znacznie ułatwia przekazywanie parametrów do podprogramów i wykonywanie obliczeń numerycznych. Aby zapewnić zgodność programową z 6800 rejestr DP jest zerowany sygnałem RESET/.

Tryby adresowania

Tryby adresowania są najsilniejszą stroną omawianego procesora. Podana niżej ich lista obejmuje również przykładowe rozkazy.

Tryb bezpośredni (ang. inherent). Dotyczy on operacji na rejestrach. Przykładem jest rozkaz MUL — pomnóż zawartość A przez zawartość B i wynik umieść w D.

Tryb natychmiastowy (ang. immediate). Dana znajduje się zaraz za kodem rozkazu. Np. LDD # $8000 — załaduj akumulator D stałą 8000_H. W tym trybie mieszczą się również rozkazy przesłań międzyrejestrowych TFR i wymiany zawartości rejestrów EXG. O tym, które rejestry biorą udział w operacji, decyduje bajt kodowy umieszczony bezpośrednio za kodem rozkazu (ang. postbyte).

Tryb rozszerzony (ang. extended). W tym trybie adres efektywny argumentu podany jest explicite po kodzie rozkazu; np. LDD 5403 — załaduj akumulator D dana 16-bitowa z komórek pamięci o adresach $5403_H/5404_H$. Ten tryb adresowania nie jest preferowany z uwagi na to, że wyklucza przemieszczalność programu.

Tryb strony zerowej (ang. direct). Adres efektywny argumentu tworzony jest przez złożenie zawartości rejestru DP i bajtu usytuowanego za kodem rozkazu. Dla przykładu, jeżeli rejestr DP zawiera 80_H, to wykonanie rozkazu LDA > 3F spowoduje załadowanie akumulatora A daną z komórki pamięci o adresie $803F_H$.

Tryb indeksowy (ang. indexed). Tryb ten ma wiele odmian. Są to:
A) Przesunięcie stałe — 5-, 8- lub 16-bitowe. Przesunięcie, w kodzie U2, dodawane jest do zawartości jednego z rejestrów indeksowych: np. LDY--32798, U — załaduj 16-bitowy rejestr Y daną umieszczoną o 32 798 komórek poniżej adresu znajdującego się w rejestrze U. Inny przykład, to CMPD 279, PCR — porównaj zawartość akumulatora D z zawartością dwu komórek pamięci, pod adresem o 279 komórek większym niż aktualna zawartość licznika rozkazów (licznik rozkazów może służyć jako dodatkowy rejestr indeksowy).

B) Przesunięcie znajduje się w jednym z rejestrów 8- lub 16-bitowych. Rozkaz LDX D,U spowoduje załadowanie rejestru X daną z komórek pamięci wyznaczonych przez dodanie zawartości D i U.

C) Przesunięcie równe jest zero — zawartość rejestru indeksowego ulega automatycznie zmianie o 1 bądź 2. Tak działają rozkazy LDB,X+ — pobierz do akumulatora B dana wskazywaną przez rejestr X, a następnie zwiększ X o 1 oraz LDB,--S — zmniejsz zawartość rejestru S o 2, a następnie załaduj dana według nowego adresu. Wprowadzono również dodatkowy poziom wyznaczania adresu (ang. indirection). Dotyczy to trybów indeksowego i rozszerzonego. W szczególności procesor wykonuje rozkazy typu: LDA [F400] — pobierz do A dana, której adres znajduje się w komórce o adresie $F400_H$; JMP [30,PCR] — skocz do adresu, który znajduje się w komórkach wyznaczonych przez sumę aktualnej zawartości licznika rozkazów i stałej 30_H czy LDD [,Y++] — pobierz do akumulatora D dana, której adres znajduje się w komórkach wskazywanych przez Y, a następnie zwiększ Y o 2.

Sterowanie w programie realizują „bliskie" i „dalekie" skoki względne (ang. branch relativ). W przypadku spełnienia warunku i wykonania skoku sterowanie przekazywane jest do adresu będącego sumą zawartości licznika rozkazów i stałej w kodzie U2, która umieszczona jest za kodem rozkazu. Rozkazy skoków bliskich (stała 8-bitowa) umożliwiają skoki na odległość —128 ... +127 komórek, a skoki dalekie (stała 16-bitowa) —32768 ... +32767.

Lista rozkazów

Procesor wykonuje 59 typów rozkazów, biorąc jednak pod uwagę wszystkie możliwe warianty adresowania,

uzyskuje się ogromną liczbę 1464 dostępnych operacji. Rozkazy kodowane są na 1, 2, 3, 4 bądź 5 bajtach. Liczba cykli maszynowych, w czasie których wykonywane są rozkazy, wynosi dwa do ośmiu (wyjątkowo tylko 20, w przypadku przerwania programowego). Przy częstotliwości zegara systemu 2 MHz dodawanie 16-bitowe trwa 2 μs, a mnożenie danych 8-bitowych 5 μs. Procesor nie dysponuje odrębnymi rozkazami wejścia/wyjścia.

Rozkazy, dotyczące akumulatorów 8-bitowych i pojedynczych komórek pamięci, obejmują operacje: dodawania, odejmowania, testowania bitów i słów, inkrementacji i dekrementacji, przesunięć i obrotów, operacje logiczne AND, OR i EXOR, mnożenie (!), zerowanie, uzupełnianie do 1 i 2, korekcję dziesiętną.

Operacje 16-bitowe, to: dodawanie, odejmowanie, porównanie, przesłania oraz przedłużenie znaku — czyli zamiana danej 8-bitowej w kodzie U2 na daną 16-bitową w tym kodzie.

Bogaty jest zestaw rozkazów operujących na stosie i działających na rejestrach indeksowych. Rozkazy przesłań na stosy użytkownika — PSHU i systemowy — PSHS przesyłają dowolne rejestry pojedynczo lub grupowo. Dotyczy to również operacji zdjęcia ze stosu — PULU i PULS. Wskaźniki stosu mogą być przesyłane (do) i z pamięci oraz brać udział w operacjach porównania. Istnieją również rozkazy obliczania adresu efektywnego — LEA (ang. Load Effective Address). Za ich pomocą można do jednego z rejestrów indeksowych załadować adres danej bez wykonywania na niej operacji. Na przykład LEAY —10,S ładuje do rejestru Y adres utworzony przez odjęcie od zawartości S stałej 10, LEAS 8,S wykonuje operację S:=S+8 (czyli „oczyszcza" 8 komórek stosu), a LEAU —D,U rezerwuje na stosie użytkownika tyle komórek, ile wynosi zawartość akumulatora D.

Rozkazy skoków względnych mogą testować bity CCR, bądź relacje zachodzące w wyniku wykonywania operacji na danych w kodach U2 lub NKB. Wszystkie skoki występują w formie „bliskiej" i „dalekiej". Ponadto istnieją rozkazy skoków do podprogramów i skoku bezwarunkowego.

Ostatnią grupę stanowią tzw. rozkazy różne. Wśród nich na uwagę zasługują trzy rozkazy przerwań programowych SWI, SWI2 i SWI3 oraz rozkaz SYNC, który pozwala na synchronizację programu ze zdarzeniami zewnętrznymi. Powoduje on wstrzymanie wykonywania programu do chwili przyjścia przerwania na linii IRQ/. Jeżeli jednak przerwanie to jest zablokowane, to program wykonywany jest dalej.

* * *

Mikroprocesor MC6809 jest produkowany w dwu wersjach. Obok 6809 występuje 6809E przeznaczony do pracy wieloprocesorowej. Dostępne są też układy towarzyszące, znacznie rozszerzające możliwości procesora. Dla przykładu układ zarządzania pamięcią MC6829 zwiększa obszar adresowy do 2Mb, a układ MC6883 spełnia rolę zegara systemu zintegrowanego ze sterownikiem pamięci dynamicznej, w tym układów 4164.

Możliwości programowe, jakie znajdujemy w mikroprocesorze 6809 nie są odległe od tych, które oferują procesory „w pełni 16-bitowe". Powyższy przegląd daje nam zatem obraz tego, czym będziemy się zajmować w przyszłości. Jak odległej, to już inne zagadnienie.

JERZY ORKISZEWSKI
Warszawa

Po olbrzymim zainteresowaniu wydrukiem kodu maszynowego programu fig-FORTH spodziewaliśmy się zalewu publikacjami nadsyłanymi przez zwolenników tego języka. Jak zwykle okazało się, że wielu chętnie korzysta, natomiast niewielu chce przekazać coś innym. Marek Czarzyński po nadesłaniu odcinka, który zamieściliśmy w 13 mikroKLANIE, i deklaracji wznowienia współpracy ponownie zamilkł. Część winy spada jednak na opóźnienie, z jakim w bieżącym roku ukazuje się INFORMATYKA. W chwili gdy oddajemy do drukarni ten numer (połowa czerwca), do rąk Czytelników trafił dopiero pierwszy numer czasopisma. Tak więc zwolennicy FORTHA nic jeszcze nie wiedzą o „kryzysie kącika" — co częściowo tłumaczy ich bierność. Ciszę przerwał Michał Choroszucha, który próbował wykorzystać procedury zamieszczone w 13 mikroKLANIE i... natknął się na wiele błędów. W poniższym materiale próbuje je sprostować, reklamując równocześnie wykonaną przez siebie modyfikację programu fig-FORTH.

Wokół języka FORTH

Poniższy tekst jest efektem analizy teoretycznej, a następnie próby uruchomienia procedur opisanych przez Marka Czarzyńskiego w 13 numerze mikroKLANU (INFORMATYKA, nr 1, 1985). Procedury były uruchamiane za pomocą zmodyfikowanej wersji programu 8080 fig-FORTH 1.1 (wersja 1.1 M), współpracującej z plikami systemu operacyjnego CP/M zamiast (fizycznie) z dyskietką, jak to występuje w wersji oryginalnej. Modyfikacja ta stanowi część pracy dyplomowej autora. Wersje źródłowe programów 8080 fig-FORTH 1.1 oraz 1.1 M są dostępne u autora [1]).

Przed opisem prawidłowych (działających) procedur przedstawione zostaną niektóre zmiany występujące w wersji 1.1 M — nie mające bezpośredniego związku z operacjami na plikach systemu CP/M. Zmodyfikowana procedura **ID.** zeruje najbardziej znaczący bit przy drukowaniu ostatniego znaku nazwy. **CREATE** „zachowuje się" prawidłowo także przy nazwach dłuższych od wartości zmiennej **WIDTH** — w przeciwieństwie do procedury oryginalnej. **VLIST** nigdy nie przekracza 79 znaków w linii — niezależnie od długości drukowanych słów. Jedynym wyjątkiem jest tu wydruk tekstu OK przez słowo **QUIT** po zakończeniu działania **VLIST.**

Nowe procedury:

?NUMBER (——d) czyta z klawiatury ciąg cyfr zakończonych znakiem powrotu karetki (CR). Wynik: w postaci liczby podwójnej na stosie parametrów oraz odpowiedniej wartości zmiennej **DPL.** Procedura wykorzystywana jest do czytania parametrów

```
: ID.
   PAD   [ HEX ]   BL   5F   FILL
   DUP   PFA   LFA   OVER   -   PAD
   SWAP   CMOVE   PAD   COUNT   1F   AND
   ( FOLLOWING APPENDED )   2DUP   +   1-
   80   TOGGLE   ( END OF MODIFICATIONS )
   TYPE   SPACE   ;

: CREATE
   -FIND   [ HEX ]
      IF   ( FOUND )
         DROP   NFA   ID.   4   MESSAGE   SPACE
      ENDIF
   HERE   DUP   C@   WIDTH   @   MIN
   ( FOLLOWING APPENDED )   2DUP   0A0
   OR   SWAP   C!   ( END OF MODIFICATIONS )
   1+   ALLOT   ( DELETED : DUP 0A0 TOGGLE )
   HERE   1-   80   TOGGLE   LATEST   ,
   CURRENT   @   !   HERE   2+   ,   ;

: VLIST
   [ HEX ]   80   OUT   !   CONTEXT   @   @
      BEGIN
         DUP   @   1F   AND   OUT   @   +   4C   >
            IF   ( LINE TOO LONG )
               CR
            ELSE
               SPACE   SPACE
            ENDIF
         DUP   ID.   PFA   LFA   @
         DUP   0=   ?TERMINAL   OR
      UNTIL   ( REPEAT UNTIL TRUE )
   SPACE   SPACE   DROP   ;

: ?NUMBER
   HERE   [ HEX ]   20   EXPECT
   HERE   #L   ERELOS   DROP
   HERE   +   #L   SWAP   C!
   +   1-   NUMBER   ;

: ?SPACES
   OUT   @   -   SPACES   ;

: TAB
   >R   OUT   @   0
      BEGIN
         2DUP   <   0=   ( )=   )
      WHILE   ( EXIT IF TRUE )
         R   +
      REPEAT   ( UNTIL TRUE DURING WHILE )
   R>   DROP   SWAP   -   SPACES   ;

: CCR
   CR   0   OUT   !   ;

: 1-
   1   -   ;

: 2-
   2   -   ;
```

Wydruk 1

[1]) Listy nadesłane do redakcji przekażemy autorowi (przyp. red.).

przez program w trybie interakcyjnym.
?SPACES (n ———) ustawia kursor w kolumnie „n"; jeśli jest to niemożliwe, to pozycja kursora pozostaje bez zmiany. Duże dodatnie wartości „n" nie są eliminowane.
TAB (n ———) tabulacja co „n' kolumn
CR w wersji 1.1 M spełnia funkcję procedury **CCR.**

Opis błędów w procedurach z nr 13 mikroKLANU

FIND — w przypadku nieobecności poszukiwanego słowa w słowniku nie przerywa działania programu, co powoduje pobranie adresu z pustego stosu i wydruku niewłaściwych tekstów.
NFA-FOR-CFA — brakuje powielenia adresu **DUP**
(błąd MSG # 1 — pusty stos). Można też zastosować jedną z procedur alternatywnych.

```
: FIND   ( CHANGED )
  -FIND
    IF
      DROP   2-
    ELSE
      0   ERROR
    ENDIF  ;

: NFA-FOR-CFA   ( CHANGED )
  3   -
    BEGIN
      1-   DUP   C@   [ HEX ]   80   )
    UNTIL  ;

: NFA-FOR-CFA'   ( ALTERNATIVES FOR  NFA-FOR-CFA )
  2+   ( PFA )   NFA
  ( OR : 3   -   -1   TRAVERSE )  ;

: NAME?   ( CHANGED ,  *WARNING* )
  ( POSSIBLE SYSTEM ERROR IF ADDRESS OUT OF MEMORY )
  DUP   NFA-FOR-CFA   DUP   C@
  1F   AND   +   3   +   =  ;

: NAME(PFA)
  2-   ( MINUS NOT TYPED IN SOURCE LISTING )
  NAME(CFA)  ;

: L2
  DUP   0   OVER   ." ADR="   .
  ( TABULATION IMPROVED )
  [ HEX ]   0C   ?SPACES   ." HEX="   DUP   .
  17   ?SPACES   DUP   ." NAME="   NAME?
    IF
      NAME(CFA)
    ELSE
      ." ????"   DROP
    ENDIF  ;
```

```
: ;S?   ( *WARNING* )
  ( UNDEFINED REPLY IF WORD DOESN'T END WITH  'SEMIS' )
  2+   DUP   @   2+   '   ;S   =  ;

: E."   ( MODIFIED ,  DELETED :  E00FF ,  EFF00 )
  1+   )R   R   +   R)
    DO   I   C@   EMIT   LOOP  ;
```

Wydruk 2

NAME? — występują błędy przy próbie wydruku słowa, w którym jest kompilator, np.:
: END [COMPILE] UNTIL ; IMMEDIATE
NAME(PFA) — brakuje znaku „—"
L2 — formowanie wydruku nie działa tak, jak to zostało opisane w „spodziewanym wydruku" (strona 19).
(.")? — brakuje znaku „?" w nazwie.
"? — można zastąpić ʙ FF AND przez C ʙ
E00FF, EFF00, E." — można znacznie uprościć. (Wydruk 2)

W procedurach **L1** oraz **LOOK-IN** należy zastąpić **CR** przez **CCR.** Pozostałe procedury nie wymagają zmian.

★ ★ ★

Stosunkowo rzadko używanym, a bardzo pożytecznym słowem FORTHA jest VOCABULARY. Przykłady jego zastosowania znalazły się w 10 mikroKLANIE (INFORMATYKA, nr 10, 1984). Niestety w opublikowanej na stronie 20 sekwencji słów znalazł się poważny błąd, który pozwala przypuszczać, że opublikowane programy nie były uruchamiane przez autora artykułu. Jak wspomniano w cytowanym artykule, stosowanie wielu słowników umożliwia pisanie programów bez konieczności rezerwowania dla nich unikalnych nazw procedur. Każda próba definiowania już istniejącej procedury powoduje ostrzeżenie: ERROR # 4: NAME NOT UNIQUE, natomiast próba kompilacji do nowej procedury słowa nie występującego w danym słowniku (ani poniżej niego, aż do końca słownika FORTH włącznie) kończy się błędem: ERROR # 0: INPUT CONVERSION. Próby ominięcia zabezpieczeń FORTHA przez zmianę wartości zmiennej CONTEXT mijają się z celem (wydruk 1), czego przyczynę łatwo znaleźć analizując działanie kompilatora :. Poniższa definicja została napisana w oparciu o [1]:

```
: : ?EXEC !CSP CURRENT ʙ CONTEXT !
  CREATE ] ;CODE IMMEDIATE [HEX]
  128 (RPP) LHLD H DCX B M MOV H DCX
  C M MOV 128 SHLD D INX E C MOV D B MOV
  NEXT (145)     JMP C;
```

Aby ominąć zabezpieczenia FORTHA zawarte w słowie ; należy odpowiednio ustawić wartość zmiennej CONTEXT dopiero po utworzeniu nagłówka definicji (wydruk 1). Przywrócenie stanu początkowego CONTEXT po zakończeniu definiowania słowa nie jest konieczne.

```
8080 FIG-FORTH 1.1 M 24K CP/M
READY
FORTH DEFINITIONS OK
: F1 ; VOCABULARY VOC-A : F3 ; VOCABULARY VOC-B OK
VOC-B DEFINITIONS OK
: B1 ; : B2 ; VOCABULARY VOC-B.B : B4 ; OK
FORTH DEFINITIONS OK
: F5 ; VOCABULARY VOC-C OK
VOC-C DEFINITIONS OK
FORTH VOC-B  ( OR :  VOC-B  ONLY ) OK
: C1 B1 B2 B4 ; B1 ?_MSG # 0
VLIST
    VOC-C   F5   VOC-B   F3   VOC-A   F1   TASK   OK
: C1          ( SETS CONTEXT TO CURRENT = VOC-C )
[ VOC-B ]     ( SETS CONTEXT TO VOC-B )
B1 B2 B4      ( COMPILES VOC-B WORDS )
[ VOC-C ]     ( RESTORES CONTEXT TO VOC-C )  ; OK
VLIST
C1      VOC-C   F5   VOC-B   F3   VOC-A   F1   TASK   OK
```

Wydruk 1

W cytowanym artykule znalazło się stwierdzenie, że zmiana wartości stałej jest niedozwolona. Nie jest to prawda; zalecany sposób zmiany wartości stałej został opisany w [2]. Używa się do tego celu słowa ' (po angielsku nazywa się je: tick) w następujący sposób:
value CONSTANT name
new-value ' name !

```
CR DECIMAL 240 LIST 241 LIST

SRC # 240
  0 ( MEMORY DUMP PRIMATIVES  **)
  1
  2 DECIMAL
  3 : (EMIT)   ( CHAR ---   EMIT DISPLAYABLE CHAR ,  ELSE .  *)
  4    127   AND   DUP   BL   (
  5       IF   DROP   46   ( "." )   ENDIF
  6    EMIT  ;
  7 : (DUMP)   ( FIRST --- NEXT   DUMP LINE  *)
  8    DUP   5   .R   ( .ADDRESS )   SPACE
  9    DUP   16   ( ADDR ADDR 16 )
 10    OVER   +   SWAP   ( ADDR 16+ADDR ADDR )
 11       DO   I   C@   3   .R   LOOP   ( ADDR )
 12    SPACE   SPACE   16   0
 13       DO   ( ADDR )   DUP   C@   (EMIT)   1+   ( ADDR+ )   LOOP
 14    CR   ( 16+ADDR )   ;
 15 --)

SRC # 241
  0 ( MEMORY DUMP WORD  **)
  1
  2 : DUMP   ( FIRST LAST ---   DUMP MEMORY )
  3    BASE   @   )R   HEX   )R   CR   ( FIRST )
  4       BEGIN
  5          (DUMP)   DUP   R   U(   0=   ?TERMINAL   OR
  6       UNTIL   ( TRUE ,  IF CURRENT )= LAST  OR  KEY PRESSED )
  7    R)   DROP   DROP   R)   BASE   !   ;   ;S
  8
OK
240 LOAD OK
VLIST
DUMP   (DUMP)   (EMIT)   TASK   OK
256 512 DUMP
  100    0 C3 85 17   0 C3 70 17   1   1   1  E B0 20 7F   0  .C...Cp.....0 .
  110   30 3F   0 3E 30 3F   0 3E 1F   0   0   0 73 20 BB 20  0?.)0?.)....s ;
  120   64 20   5   0 20 B3 30 3F 30 3F   0   0 21 82 3F 7E  d .. 30?0?..!.?~
  130   B9 C2 3D   1 23 7E B8 C2 3D   1   0   0   0   A   3 6F  9B=.#~8B=.......o
  140   C3 48   1 D5 E5   A   3 6F   A   3 67 5E 23 56 E5 E9  CH.Ue...o...g^#Vki
  150   83 4C 49 D4   0   0 58   1   A   3 6F   A   3 67 C3 44  .LIT...X....o...gCD
  160    1 87 45 58 45 43 55 54 C5 50   1 6D   1 E1 C3 4B  ..EXECUTEP.m.aCK
  170    1 86 42 52 41 4E 43 C8 61   1 7C   1 60 69 5E 23  ..BRANCHa.|.`i^#
  180   56 2B 19 4D 44 C3 45   1 87 30 42 52 41 4E 43 C8  V+.MDCE..0BRANCH
  190   71   1 94   1 E1 7D B4 CA 7C   1   3   3 C3 45   1 84  q....a}4J|...CE..
  1A0   28 44 4F A9 88   1 A8   1 2A 86 3F 2B 2B 2B 2B 22  (DO)..(.*.?++++"
  1B0   86 3F D1 73 23 72 D1 23 73 23 72 C3 45   1 86 28  .?Qs#rQ#s#rCE..(
  1C0   4C 4F 4F 50 A9 9F   1 C9   1 11   1   0 2A B6 3F 7E  LOOP)..I....*.?~
  1D0   83 77 5F 23 7E 8A 77 23 14 15 57 FA E6   1 7B 96  .w_#~.w#..Wz.{.
  1E0   7A 23 9E C3 EB   1 7E 93 23 7E 9A FA 7C   1 23 22  z#.Ck.~.#~.z|.#"
  1F0   86 3F   3   3 C3 45   1 87 28 2B 4C 4F 4F 50 A9 BE  .?..CE..(+LOOP))
OK
BYE
```

Wydruk 2

Poniżej zostaną przedstawione słowa służące do wyprowadzania zawartości pamięci [1]). Drukowane są kody szesnastkowe poszczególnych bajtów pamięci oraz ich odpowiedniki według kodu ASCII. Procedury można łatwo modyfikować dla dostosowania ich do innych długości linii na ekranie, można również drukować słowa pamięci, a nie poszczególne bajty (należy użyć konstrukcji ... 2 +LOOP).

Dla dociekliwych: znak o kodzie HEX 7F (ang. delete) nie jest obrazowany przez drukarkę, co można uwzględnić w słowie ⟨EMIT⟩. Jeśli ktoś chce oglądać obszar pamięci poza pamięcią zajętą przez FORTHA, konkretnie powyżej HEX 8000, to lepiej niech zmieni ósmą linię kadru 240 (wydruk 2) na:

DUP 0 5 D.R SPACE

W przeciwnym wypadku adresy będą podawane jako liczby ujemne.

Długie wydruki można przerwać przez wciśnięcie dowolnego klawisza (radzę stosować znak „delete", gdyż nie daje to żadnych widocznych efektów w następnej linii ekranu). W słowie (DUMP) zaprezentowane zostały dwie spośród trzech najczęściej stosowanych technik programowania pętli DO:

(start count) OVER + SWAP

DO ... I (start...start+count—1) ... LOOP ( )
(start count 0)

DO ... 1+ ... LOOP (start+count)
(start count 0)

DO ... DUP I + (start start+I) ... LOOP (start)

W nawiasach podano stan istotnych liczb na stosie parametrów zgodnie z [1], czyli: wierzchołek stosu po prawej stronie, jeśli występuje ciąg vvv ——— yyy, to znaczy, że vvv jest stosem przed, a yyy — po wykonaniu definicji; brak liczby oznacza stos pusty (w stosunku do sytuacji określonej jako poprzednia).

```
240 LIST
SRC ⊕ 240
  0 ( FORTH79 WORDS :  DEPTH  PICK ;  !HEAP  **)
  1 : DEPTH   ( --- STACK_DEPTH  *)
  2   S0  @  SP@  -  2-  2  /  ;
  3 : (HEAP)   ( i-1 --- ADDR_Ni  *)
  4   DEPTH  2-  DUP  0(  9  ?ERROR
  5   OVER  U(  9  ?ERROR
  6   1+  DUP  +  SP@  +  ;
  7 : PICK   ( i --- Ni  *)
  8   1-  (HEAP)  @  ;
  9 : !HEAP   ( value i ---   MODIFY STACK  *)
 10   DUP  0=  9  ?ERROR
 11   (HEAP)  !  ;
 12 ;S
 13
 14
 15
OK
240 LOAD OK
VLIST
!HEAP  PICK  (HEAP)  DEPTH  TASK  OK
1 3 2 4 5 0 DEPTH . 6 OK
9 4 !HEAP 7 2 !HEAP OK
. . . . . 0 7 4 9 3 1 OK
5 4 8 2 1 3 PICK . . . . . 8 1 2 8 4 5 OK
BYE
```

Wydruk 3

Wydruk 3 przedstawia przykładową implementację w języku fig-FORTH 1.1 słów języka FORTH79: **DEPTH** i **PICK** (patrz mikroKLAN 2, INFORMATYKA nr 2, 1984) oraz nowe słowo **!HEAP** służące do modyfikacji stosu względem jego wierzchołka.

W następnym odcinku opisane zostaną standardowe komunikaty FORTHA, słowo **ROLL**, a także kolejne błędy fig-FORTHA 1.1 i sposób ich usunięcia.

MICHAŁ CHOROSZUCHA

[1]) Standard FORTH79 przewiduje podawanie zamiast parametru LAST w słowie DUMP długości wyświetlanego obszaru w bajtach. Pozwala to na oglądanie słów FORTHA za pomocą następującej sekwencji:
' name length DUMP
dla obejrzenia pola parametrów, lub:
' name NFA length DUMP
dla całej definicji. Dla otrzymania takiej implementacji słowa DUMP należy zmienić trzecią linię kadru 241 (wydruk 2) na:
BASE @ >R HEX OVER + >R CR

LITERATURA

[1] fig-FORTH INSTALLATION MANUAL. GLOSSARY, MODEL, EDITOR. Release 1 with compiler security and variable length names — by William F. RAGSDALE November 1980, provided through the courtesy of the FORTH INTEREST GROUP Po Box 1105, San Carlos, CA 94070, USA

[2] FORTH, An Application-Oriented Language, REFERENCE MANUAL. Writer: Elizabeth D. Rather, Editors: Carolyn A. Rosenberg, Sue Linstrot. FORTH, Inc. 2309 Pacific Coast Highway, Hermosa Beach, CA 90254, USA, 1979—1980.

Samotesty

IV/B. Rzetelność danych

Samotesty, opracowane i opublikowane przez Association for Computing Machinery *), nie stanowią standardu egzaminacyjnego i są przeznaczone wyłącznie do samodzielnego sprawdzenia własnej wiedzy.

PYTANIA

9. Które z wymienionych niżej pojęć **nie ma** bezpośredniego związku z badaniem poprawności danych w zbiorze wejściowym?
   a) suma kontrolna (ang. check sum)
   b) liczba przerwań programu (ang. interruption count)
   c) suma umowna (ang. hash total)
   d) liczba rekordów (ang. record count)
10. Wymień kilka testów weryfikacyjnych stosowanych w badaniu rzetelności (ang. integrity) danych wejściowych.
11. Uzupełnij poniższe zdanie najwłaściwszym z wymienionych określeń: „Dla celów weryfikacji można wyliczyć ..... — z dołączonych przy nazwiskach pracowników numerów identyfikacyjnych — w fazie poprzedzającej właściwe sporządzanie listy płac".
    a) liczbę rekordów (ang. record count)
    b) sumę umowną (ang. hash total)
    c) cyfrę kontrolną (ang. check digit)
    d) saldo poziome (ang. crossfoot total)
12. Uzupełnij właściwą liczbą następujące zdanie: „Rozszerzając kod numeryczny o dodatkową cyfrę, wyrażającą sumę cyfr modulo 11, zapewniamy sobie wykrywanie ...% ogółu błędów powodowanych przestawieniem dwu cyfr".
    a) 0 b) 50 c) 90 d) 100
13. Do jakich z wymienionych niżej celów można w rachunkowości wykorzystywać tzw. pliki kontrolne (ang. audit files) albo dzienniki transakcyjne (ang. journals)?

*) Self-Assessment Procedure IV. Program Development Tools and Methods. Part B. Data Integrity. Communications of the Association for Computing Machinery, vol. 21, No. 2, February 1978, pp. 112—114.

a) jako pliki wtórne (ang. backups) umożliwiające odtwarzanie (ang. recorvering) danych utraconych lub zniekształconych
b) jako historyczny materiał źródłowy do wykrywania przypadków naruszenia poufności danych (ang. security violation)
c) jako katalog dla stale dostępnych plików roboczych użytkownika (ang. on-line user files)
d) jako indeks ewidencji prowadzonych w poszczególnych agendach przedsiębiorstwa (ang. industry journals).

ROZWIĄZANIA

a-b; 13 — d; 12 — b; 11 — przypisy; 10 — b-9.

LITERATURA I KOMENTARZE:

ad 9. Cushing B.: Accounting Information Systems and Business Organizations. Addison-Wesley, p. 286
Sanders D.: Computers in Business. McGraw-Hill pp. 531—532.

ad 10. W zapewnianiu rzetelności danych wejściowych najbardziej przydatnymi są testy dopuszczalności (ang. validity tests) dla:
— kodów wejściowych
— znaków
— rozmiarów pól
— typów transakcji
— kombinacji wspólnych cech danych wejściowych
— opuszczeń danych
— uporządkowań danych wejściowych
— przedziałów wartości danych lub sensowności danych.
Por. Dawis C.: Computer Data Processing, IInd edition, McGraw-Hill, pp. 432—434.

ad 11. Clark F., Gale R. and Gray R.: Business Systems and Data Processing Procedures. Prentice-Hall, pp. 207—210
Kindred A.: Data Systems and Management. Prentice-Hall, pp. 249—252
Davis, ibid. pp. 431—438.

ad 12. Burch J., Strater F.: Information Systems: Theory and Practice. Hamilton Publ. p. 166.

ad 13. Myers G.: Software Reliability — Principles and Practices. Willey, New York 1976, pp. 256—257.
Przekład polski Stanisława Matwina: Projektowanie niezawodnego oprogramowania. WNT, Warszawa 1980, pp. 236—237.

Adaptowali z angielskiego:
ADAM B. EMPACHER
JERZY L. ROSSOWSKI

Ze świata

Producenci komputerów 8-bitowych w opałach

Przełom roku 1984—1985 bardzo wyraźnie wstrząsnął układem sił na brytyjskim rynku komputerowym. Najpierw, późną jesienią 1984 r. pojawiły się doniesienia o spadku zakupów mikrokomputerów 8-bitowych. Potem, gdy tylko skończył się sezon na prezenty pod choinkę, nastąpiły drastyczne obniżki cen. Np. SINCLAIR obniżył cenę modelu ZX SPECTRUM + ze 179 na 129 funtów, podobnie potraktował swoje mikrokomputery ACORN (BBC, ELECTRON). Styczniowy zastój w sklepach wzmagał tylko pogłoski o problemach czołowych firm, dla których rok 1984 był dużo mniej udany niż 1983. Wreszcie w lutym br. wybuchła bomba: najwięksi producenci brytyjscy, tj. ACORN i SINCLAIR RESEARCH znaleźli się w poważnych kłopotach — konkretnym ich przejawem był gwałtowny spadek wartości akcji tych firm na giełdzie. Po pierwszym chwilowym zamieszaniu okazało się, że SINCLAIR stoi całkiem dobrze, natomiast ACORN zniknął z rynku i z giełdy. Było to dość zaskakujące, bo o ile Sir Clive Sinclair jest znany z rozmaitych ekstrawaganckich pomysłów i pewnego woluntaryzmu w prowadzeniu firmy, o tyle pozycja ACORNA wydawała się dość stabilna, m.in. dzięki bardzo silnej pozycji na rynku edukacyjnym (85% mikrokomputerów zainstalowanych w brytyjskich szkołach podstawowych i średnich zostało wyprodukowanych przez firmę ACORN — są to głównie modele BBC).

Wszystko jednak świadczy o szybszym niż oczekiwano nasyceniu rynku mikrokomputerami 8-bitowymi, traktowanymi jako zabawka. Użytkownicy, którzy poznali już możliwości mikrokomputerów osobistych, mają teraz większe apetyty i szukają raczej zestawów profesjonalnych z dobrą grafiką, z rozsądnym pakietem oprogramowania, z dyskietkami. Podobnie wzrosły wymagania szkół — dotyczy to nie tylko sprzętu, ale i programów edukacyjnych.

Obawy, że ACORN zbankrutował, dość szybko okazały się nieuzasadnione. Firma straciła jednak swoją niezależność i została praktycznie przyjęta przez włoskiego giganta OLIVETTI. W marcu i kwietniu br. zaczęły wychodzić na jaw konkretne fakty. Okazało się, że ACORN pod koniec ubiegłego roku miał 48 mln funtów długu przy ok. 10 mln „wolnego" kapitału. OLIVETTI wniosła 12 mln funtów w postaci kapitału obrotowego i otrzymała udziały wynoszące co najmniej 49,3% (według niektórych opinii OLIVETTI całkowicie przejęła kontrolę nad ACORNEM, dysponując 50,1% udziałów). Plan ratowania firmy przedstawiony przez OLIVETTI zyskał akceptacje kół finansowych, m.in. uzgodniono spłatę ok. 20 mln funtów w ciągu roku, a akcje ACORNA znów pojawiły się na giełdzie. O skali problemu może też świadczyć fakt, że w trakcie negocjacji prezes OLIVETTI, Carlo de Benedetti, spotkał się nie tylko z brytyjskim ministrem ds. informatyki, ale także z Normanem Tebbitem, ministrem handlu i przemysłu.

Wymienia się różne powody załamania się ACORNA, takie jak: duże straty poniesione wskutek nieudanej próby wejścia na rynek amerykański, kontynuowanie produkcji niezbyt udanych modeli mikrokomputerów, słaby dział marketingowy firmy. Jednak przeważa chyba opinia, że wszystkie te błędy miały swe źródło w nie najlepszym zarządzaniu, pośrednio wynikającym z faktu, że praktycznie firma była własnością założycieli — Chrisa Curry i Hermana Hausera, którzy dysponowali ok. 85% udziałów. Uważa się, że wraz ze wzrostem rozmiarów przedsiębiorstwa nie nastąpiły odpowiednie zmiany w metodach zarządzania. W efekcie — pomimo takich atutów jak dobre laboratorium rozwojowe i utalentowany zespół konstruktorów — próba rozwoju firmy z poziomu dużego przedsiębiorstwa do poziomu koncernu o zasięgu międzynarodowym zakończyła się katastrofą. Mimo utraty niezależności oraz poważnej restrukturalizacji firmy, narzuconej przez OLIVETTI, przyszłość ACORNA nie wydaje się taka zła. W szczególności, gdy wiadomo, że BBC, tj. telewizja brytyjska, podtrzymuje kontrakt na budowę mikrokomputerów typu BBC, co oznacza utrzymanie podstawowego atutu firmy.

W porównaniu z tą sytuacją pozycja firmy SINCLAIR wydaje się być silna. Obserwatorzy zwracają jednak uwagę, że ze względu na prywatny charakter tej firmy, możliwości tuszowania rozmaitych nieprzyjemnych faktów są tu większe niż np. spółek akcyjnych. Znaczny spadek zysków nie dał się wprawdzie ukryć, ale ile w rzeczywistości wynosiły straty poniesione na różnych konkretnych przedsięwzięciach — tego nikt nie wie. Niemniej szacuje się, że SINCLAIR na przełomie lat 1984—1985 poniósł straty o łącznej wysokości ok. 7,5 mln funtów. Wydaje się, że produkcja starszych modeli na rynki zachodnie została definitywnie wstrzymana i obecnie w sklepach widać tam praktycznie tylko modele ZX SPECTRUM + oraz QL.

Podczas pierwszych miesięcy br. pozycja firmy nie uległa zmianie, a prasa informatyczna i ogólna wyraża się o ostatnich przedsięwzięciach Sir Clive'a raczej z przekąsem. Tak np. w styczniu z wielką reklamą wprowadzono na rynek elektryczny trójkołowiec o nazwie C5 i koszcie takim jak mikrokomputera QL, tj. 399 funtów. Mimo rewelacyjnej ceny, parametry tego odkrytego, jednoosobowego pojazdu (zasięg 20 mil, prędkość max 15 mil/h, a przede wszystkim niska pozycja fotela kierowcy — niewiele lepsza niż w gokarcie) nie przysporzyły mu wielu zwolenników — czytaj ryzykantów. Produkcja zaplanowana na tysiące sztuk miesięcznie została najpierw wstrzymana, a obecnie utrzymuje się na poziomie 100 egzemplarzy miesięcznie. Po okresie podniecenia nowością, nawet nastolatki, wynajmowane do jeżdżenia trójkołowcem przez cały dzień po Londynie, nie mogą zapewnić Sir Sinclairowi większej liczby klientów. Spodziewanych zysków nie widać więc, a tymczasem koszty rozwojowe trójkołowca są szacowane na ok. 7 mln funtów!

Nie wiadomo też jak potoczą się losy innej nowości — miniaturowego telewizora, który ma wymiary podobne do wymiarów kalkulatorów HEWLETTA-PACKARDA, jest jednak trochę grubszy i oczywiście cięższy. Telewizorek można ująć w dłoń i z pewnością jest łatwo przenośny, ale oczywiście wadą są rozmiary ekranu. Dla sprawiedliwości dodajemy, że zastosowano płaski ekran. Technologia może więc być opłacalna w przyszłości, gdyż tego typu ekrany stosuje się już w mikrokomputerach przenośnych.

O ile nawet telewizorek okaże się sukcesem, to i tak losy firmy SINCLAIR ważą się gdzie indziej. Przede wszystkim model QL, po początkowym rozczarowaniu firmy niskim poziomem sprzedaży, szczególnie w pierwszej połowie 1984 r., zyskuje powoli solidne oprogramowanie. W ostatnich kilku miesiącach pojawiły się doniesienia o wprowadzeniu na rynek kompilatorów języków C oraz PROLOG, przeznaczonych dla tego modelu. Oba języki mają czysto profesjonalny charakter, a znaczenie PROLOGU bardzo wzrasta ze względu na zastosowania w tworzeniu systemów ekspertowych. Za wcześnie jednak mówić o zdecydowanym sukcesie tego minikomputera, gdyż w odróżnieniu od rynku zabawowo-hobbystycznego na rynku minikomputerów profesjonalnych panuje dużo ostrzejsza konkurencja, a potencjalni przeciwnicy dysponują często znacznie większym kapitałem. Spadek sprzedaży mikrokomputerów 8-bitowych na rynku zachodnim skłania SINCLAIRA do ekspansji na kraje socjalistyczne i kraje trzeciego świata. Wiadomo, że firma ma już podpisane umowy z ChRL, negocjuje z NRD i ZSRR, a być może także z CSRS i Węgrami.

Stara prawda o tym, że kto stoi w miejscu, ten się cofa, w przypadku rynku mikrokomputerowego oznacza pochłonięcie przez konkurencję. Prawdziwe nadzieje rozwoju wiąże więc SINCLAIR z opanowaniem nowej technologii. Chodzi o masową produkcję układów waflowych (WSI — Wafer Scale of Integration). Układ taki obejmuje cały, kilkucentymetrowej średnicy krążek kryształu, na którym znajduje się wiele układów scalonych. Idea jest dość stara, bowiem datuje się z lat sześćdziesiątych i polega na takim zaprojektowaniu układu, aby niepotrzebne było rozcinanie całego „wafla" na poszczególne części. Oczywistym problemem są wady produkcyjne — cięcie „wafla" na poszczególne układy pozwala po prostu odrzucać niesprawne. SINCLAIR zamierza produkować tą techniką układy pamięciowe o pojemności 7 Mbitów. Jak na razie wygląda na to, że główną przeszkodą w realizacji ambitnego projektu jest znalezienie kapitału zakładowego w wysokości 50 mln funtów.

Opracował na podstawie prasy brytyjskiej

RYSZARD K. KOTT

Firmy brytyjskie wchodzą na rynek krajów socjalistycznych

Brytyjskie firmy mikrokomputerowe wykazują duże zainteresowanie rynkiem krajów socjalistycznych. Wiąże się to przede wszystkim z planami tych krajów, zmierzającymi do wprowadzenia techniki mikrokomputerowej i komputerów osobistych do szkół średnich. NRD zamierza zainstalować przed 1990 r. ok. 20 tys. mikrokomputerów w szkołach średnich. Podobne zamiary ma ZSRR, gdzie mówi się, że do tego samego roku każda szkoła średnia powinna dysponować zestawem 20 mikrokomputerów. Wiele krajów zakupiło — tytułem próby — rozmaite ilości tego typu mikrokomputerów. INFORMATYKA donosiła już o sytuacji na rynku węgierskim. Wiadomo również, że CSRS zakupiła w połowie zeszłego roku kilkaset egzemplarzy ZX81. Tak np. ACORN sprzedał moskiewskiemu Instytutowi Pedagogicznemu mikrokomputery BBC za ok. 20 tys. funtów i wierzy, że za tym nastąpią dalsze, większe zamówienia.

Obecnie nadeszła pora negocjowania poważnych kontraktów, bowiem kraje RWPG przygotowują swoje plany 5-letnie na okres 1986—1990. Firmy zachodnie wiedzą, że jest to najlepsza pora do zawierania długookresowych kontraktów i dlatego przejawiają w tym kierunku dużą aktywność. Na przykład, firma SINCLAIR RESEARCH bardzo poważnie potraktowała styczniowy pierwszy pokaz komputerów osobistych w Moskwie, a potem Międzynarodowe Targi Lipskie. Jeśli chodzi o firmy brytyjskie, w tym wyścigu po wielkie zamówienia przewodzą SINCLAIR, ACORN (mimo przejęcia przez OLIVETTI) oraz ICL. Do rozmów na wysokim szczeblu doszło także w NRD. Ich przedmiotem jest podjęcie produkcji modelu ZX SPECTRUM, a być może również modelu QL (chociaż na QL nadal obowiązują restrykcje eksportowe).

Oczywiście wśród krajów RWPG największy potencjalny rynek przedstawia Związek Radziecki. Ostatnio w Wielkiej Brytanii W. Monachow i W. Bykow, odpowiedzialni za radziecki program mikrokomputeryzacji szkół, przeprowadzili wstępne negocjacje z SINCLAIR RESEARCH (ZX SPECTRUM+) oraz z ACORNEM (model BBC). Przedmiotem negocjacji było m. in. opracowanie modeli z klawiaturą o alfabecie rosyjskim oraz odpowiednich podręczników. Rozmowy te dotyczyły uzyskania licencji. Natomiast firma ICL, tradycyjnie utrzymująca dobre stosunki handlowe z krajami socjalistycznymi, prowadzi negocjacje dotyczące budowy całych zakładów wytwarzających mikrokomputery. Kontrakt ten znajduje się już w końcowej fazie negocjacji; wiadomo bowiem, że firma uzyskała już zezwolenie brytyjskiego Ministerstwa Handlu i Przemysłu na zawarcie transakcji w wysokości do 100 mln funtów. Obie strony korzystają z niedawno zniesionego zakazu eksportu, ponieważ umowa dotyczy mikrokomputera ICL, PC, w którym zastosowano mikroprocesor 16-bitowy, INTEL 8088.

Poza krajami RWPG olbrzymim zainteresowaniem firm mikrokomputerowych cieszy się też ChRL. SINCLAIR zawarł już umowę licencyjną dotyczącą montażu mikrokomputerów SPECTRUM i ma także nadzieję na znaczne zakupy przez ten kraj gotowych egzemplarzy. Jeśli chodzi o inne firmy o zasięgu światowym, to podobne umowy z ChRL zawarły również SANYO (Japonia) oraz COMMODORE (USA). Uważa się, że rynek chiński potencjalnie może być nawet większy od radzieckiego ze względu na występującą tu znacznie większą lukę technologiczną.

Na marginesie tych informacji należy zauważyć, że istnieje również bardzo duże zainteresowanie technologią produkcji mikrokomputerów 16-bitowych, a zwłaszcza 32-bitowych, opartych na takich procesorach, jak INTEL 8086 oraz MOTOROLA 68000 i 68020. Jednakże mikroprocesory te są objęte tzw. drugim poziomem restrykcji, tj. decyzja o ewentualnej licencji

może być podjęta tylko na szczeblu rządowym. O tym, że restrykcje traktuje się poważnie, świadczą pierwsze wyroki sądowe — m. in. Brytyjczyk, M. Ludlam został skazany na dwa lata więzienia za eksport do Bułgarii minikomputerów VAX i PDP oraz systemów mikrokomputerowych SYSTIME 300. Niemniej sprawa wywołuje poważne kontrowersje, szczególnie w Europie, gdzie koła przemysłowe i handlowe uważają restrykcje narzucone przez własne rządy — pod naciskiem USA — za przesadne, albo wręcz za perfidnie maskowany protekcjonizm rządu USA, chroniący interesy firm amerykańskich. Jednak wspomniany drugi poziom restrykcji może stanowić furtkę dla zainteresowanych rządów.

na podstawie prasy brytyjskiej

Opracował RYSZARD K. KOTT

Targi Hanowerskie

W środowisku informatycznym stosunkowo mało jest znany fakt, że od kilku lat największą w Europie ekspozycją sprzętu komputerowego i związanej z nim infrastruktury jest CeBIT (Centrum für Büro — und Informationstechnik — Centrum Techniki Biurowej i Informacyjnej).

CeBIT odbywa się co roku w kwietniu, w ramach Targów Hanowerskich (RFN), w specjalnie na ten cel wybudowanej w 1970 r. olbrzymiej hali o powierzchni ok. 130 000 m² (największa na świecie hala wystawowa!). Ze względu na szybkość rozwoju branży, zwłaszcza w wyniku rewolucji mikroprocesorowej, liczba wystawców oraz zapotrzebowanie na powierzchnię wystawową w ciągu ostatnich lat znacznie wzrosły, powodując konieczność sukcesywnego rozszerzenia CeBIT o następne hale. Ostatnia ekspozycja obejmowała pięć hal połączonych w jeden kompleks o łącznej powierzchni 180 000 m². O rozmiarach CeBIT najlepiej świadczy jego porównanie ze słynnym paryskim salonem informatycznym SICOB, który dysponuje tylko ok. 83 000 m² powierzchni wystawowej.

Mimo takich rozmiarów powierzchni CeBIT, w ostatnich trzech latach nie można było już pokryć zapotrzebowania na ok. 60 000 m² dodatkowej powierzchni, zgłoszonego przez nowych i znaczną liczbę dotychczasowych wystawców. Oznaczało to w br. konieczność odmowy przydzielenia stoisk dla ok. 200 firm, a także znacznego ograniczenia powierzchni dla ok. 300 wystawców. Sytuacja taka, jak również powstałe przeciążenie infrastruktury miasta Hanower (750 tys. zwiedzających w ciągu 8 dni!), skłoniło kierownictwo Targów do wydzielenia CeBIT od 1986 r. w samodzielną imprezę. Po raz pierwszy odbędzie się on w okresie od 12 do 19 marca 1986 r. Ekspozycja zwiększy się wówczas o dalsze cztery wielkie hale wystawowe, co zapewni nie tylko całkowite zlikwidowanie istniejących kolejek firm oczekujących i przyjęcie dalszych wystawców, ale również spowoduje radykalne rozgęszczenie całej ekspozycji, i co najważniejsze, umożliwi jej grupowanie tematyczne.

Grupowanie takie, to zasadniczy przełom w dotychczasowej koncepcji organizacyjnej ekspozycji, która — w miarę wzrostu rozmiarów imprezy — była coraz bardziej krytykowana właśnie ze względu na całkowitą przypadkowość lokalizacji poszczególnych rodzajów produktów czy usług, wbrew szybko pogłębiającej się wewnątrzbranżowej specjalizacji informatyki. Brak tematycznego pogrupowania stoisk stanowi bowiem olbrzymie utrudnienie dla zwiedzających, spośród których każdy — ze względu na osobistą specjalizację — jest zainteresowany tylko określoną problematyką. W takiej sytuacji istotną pomocą dla zwiedzającego był znakomicie funkcjonujący i bardzo łatwo dostępny komputerowy system informowania EBI (Elektronische Besucher Information — Elektroniczne Informowanie Zwiedzających). Zapewniał on każdemu indywidualny przewodnik w postaci natychmiastowego wydruku szczegółowych odpowiedzi na zapytania dotyczące lokalizacji (numerów stoisk) określonych produktów oraz firm.

O kompleksowości imprezy świadczy również fakt, że tegoroczny CeBIT, oprócz ekspozycji ponad 1200 firm, tradycyjnie obejmował cykl konferencji „CeBIT Forum'85", tym razem pod hasłem „Informacja jako czynnik wytwórczy — technika informacyjna łącznikiem biura z produkcją". Cykl ten obejmował 16 sesji, podczas których wygłoszono kilkadziesiąt referatów i przeprowadzono wiele dyskusji na aktualne tematy współczesnej informatyki.

Z oczywistych względów informatyka wyszła na Targach również poza ramy CeBIT, ujawniając się, a nawet dominując jako symbol nowoczesności, w wielu halach ekspozycji przemysłu elektrotechnicznego i maszynowego. Szczególnie widoczne było to w hali nr 12 „Mikroelektronika", której ekspozycja prawie w całości kwalifikowała się do włączenia w tematykę CeBIT.

Przechodząc do syntetycznego scharakteryzowania tegorocznej ekspozycji, należy stwierdzić, że oprócz pełnej prezentacji światowej oferty nowości sprzętu komputerowego, CeBIT pokazuje również całą niezwykle zróżnicowaną infrastrukturę współczesnej informatyki. W infrastrukturze tej przybysza z Polski szczególnie szokuje ogrom oferty wydawców czasopism i książek informatycznych. Obejmuje ona dosłownie setki pozycji, zwłaszcza dostosowanych do poziomu percepcji masowego użytkownika sprzętu mikrokomputerowego. Wydaje się, że w tej dziedzinie nasze zapóźnienie jest większe niż w zakresie sprzętu, oprogramowania czy zastosowań, ze względu na kompletny brak na naszym rynku wydawniczym popularyzotorów problematyki informatycznej.

Wspomniane na wstępie rozmiary ekspozycji CeBIT oraz ograniczenia w objętości INFORMATYKI nie pozwalają na systematyczne przedsta-

wianie wszystkich nowości. Nowości tych ze względu na coraz ostrzejszą walkę konkurencyjną producentów sprzętu jest wprawdzie bardzo dużo, ale wskutek wyrównanego poziomu technologii i pewnej stabilizacji rozwoju trudno jest wskazać pozycje stanowiące rzeczywisty przełom, jak to było jeszcze 2—3 lata temu.

Najbardziej uderzającym zjawiskiem obecnego stadium rozwoju sprzętu jest wyraźna dominacja w ofercie targowej komputerów osobistych, przystosowanych zarówno do obsługi indywidualnych stanowisk pracy, jak i do pełnienia funkcji inteligentnych terminali. Dominacja ta została dodatkowo wzmocniona przez tendencję do rozszerzania możliwości funkcjonalnych sprzętu zaliczanego do kategorii komputerów domowych i tym samym zacierania się istniejących dotąd różnic.

Widoczne w każdej ekspozycji, jeszcze przed kilku laty, duże systemy komputerowe praktycznie zniknęły z tegorocznych stoisk CeBIT, mimo że jako podstawowe elementy sieci komputerowych i systemów wielodostępnych są one nadal oferowane przez tradycyjnych producentów tej kategorii sprzętu (IBM, SPERRY, CDC, HONEYWELL, SIEMENS, ICL).

Jeśli chodzi o mikrokomputery, to rzuca się w oczy postępujące ujednolicenie rozwiązań konstrukcyjnych oraz wyrównanie parametrów eksploatacyjnych. Zaostrza się walka konkurencyjna pomiędzy tradycyjną czołówką producentów, spośród których wszyscy już wyszli z szeroką ofertą sprzętu mikrokomputerowego, a dziesiątkami małych, często zupełnie nowych firm próbujących, często z powodzeniem, szans dzięki swej prężności działania i niewiarygodnej chłonności rynku światowego.

Uderzającą cechą tej części ekspozycji było coraz liczniejsze wejście zwiedzających do wnętrza stoisk, w celu samodzielnego testowania sprzętu mikrokomputerowego i jego oprogramowania. Jest to całkowicie odmienny obraz w stosunku do tego, jaki można było obserwować przed kilku laty, gdy zwiedzający w skupieniu i z dużym respektem słuchali wyjaśnień firmowych prezenterów oraz z odpowiedniej odległości obserwowali czynności profesjonalnych operatorów sprzętu.

Należy spodziewać się, że ze wspomnianej ostrej walki konkurencyjnej na rynku mikrokomputerowym wyjdą zwycięsko „wielcy" producenci, którzy mimo niewątpliwie mniejszej operatywności (charakterystycznej dla dużych jednostek organizacyjnych!) dysponują olbrzymim zapleczem badawczo-rozwojowym, większymi kapitałami, i co najważniejsze, doświadczonym aparatem obsługi użytkowników. Potwierdzeniem takiego kierunku rozwoju są powtarzające się informacje o natrafieniu na trudności zbytu przez niektórych, upojonych jeszcze wczorajszymi sukcesami, „młodych" producentów (APPLE, SINCLAIR).

Samoistnie postępująca standaryzacja rozwiązań sprzętowych (od układów podstawowych do urządzeń peryferyjnych) przenosi się obecnie do sfery oprogramowania, które jest znacznie mniej podatne na tego rodzaju działania. Uwidacznia się to najlepiej w wysiłkach praktycznie wszystkich producentów, zmierzających do wykorzystania możliwości eksploatacyjnych, jakie stwarzają dominujące już na rynku standardowe systemy operacyjne UNIX oraz MS-DOS. W przypadku systemu UNIX chodzi o pozyskanie dla sprzętu możliwości wielodostępu warunkującego rozwój sieci komputerowych, natomiast MS-DOS pozwala użytkownikom komputerów osobistych wykorzystać bardzo bogate już — w tej kategorii sprzętu — oprogramowanie użytkowe.

Innym, najbardziej chyba skutecznie oddziałującym stymulatorem standaryzacji rozwiązań jest firma IBM, której dominująca pozycja rynkowa uwidoczniła się również w zakresie sprzętu mikrokomputerowego. Trudno było spotkać stoisko w tej kategorii sprzętu, które w promocji swych wyrobów nie używałoby hasła „zgodny z IBM" (IBM compatible). Jest to, moim zdaniem, najlepsze potwierdzenie wysuwanej coraz częściej tezy o ponownym, analogicznym do zjawisk sprzed dwudziestu lat, powrocie do procesów koncentracji produkcji sprzętu.

WŁADYSŁAW KLEPACZ

Bezpośrednie systemy informacyjne

Nakładem Państwowego Wydawnictwa Ekonomicznego w serii „Informatyka w praktyce" ukazała się najnowsza pozycja — pt. „Bezpośrednie systemy informacyjne" [1]). Podtytuł „Z doświadczeń projektanta i użytkownika" jest zapowiedzią łącznej prezentacji obu sfer działalności, co rzadko się zdarza w piśmiennictwie z dziedziny informatyki.

W pierwszej chwili może zastanowić sam tytuł, czy książka dotyczy systemów informatycznych (struktur informacyjnych), takich, jakie rozpatruje się np. w teorii organizacji i zarządzania, czy raczej określonych rodzajów systemów informatycznych, na co wskazuje przymiotnik „bezpośrednie".

Dyskusję nad tytułem można pominąć, przytaczając definicję autora: **„bezpośredni system informacyjny jest to system teleinformatyczny z bezpośrednim dostępem do informacji (danych), uzyskiwanych za pośrednictwem łączy telekomunikacyjnych (w trybie on-line)".**

Mówiąc inaczej, książka dotyczy systemów o działaniu bezpośrednim. A jej novum to przede wszystkim prezentacja doświadczeń zdobytych podczas projektowania i wdrażania pierwszego w Polsce tego rodzaju systemu. Warto podkreślić, że publikacja zawiera nie tylko przejrzyście i chronologicznie omówione problemy projektowania systemów w działaniu bezpośrednim, lecz też — co na pewno nie mniej istotne — opis trudności i kłopotów, z jakimi może się spotkać projektant lub użytkownik.

Książka składa się z sześciu rozdziałów i przypisów, które właściwie są trzema oddzielnymi aneksami. Na przykład, w drugim z tych przypisów zawarto syntetyczny opis pierwszego w kraju łącza teleprzetwarzania danych między Poznaniem a Warszawą, zrealizowanego całkowicie na sprzęcie polskiej produkcji.

Zaproponowano, by budowę systemu o działaniu bezpośrednim, niemalże od początku, już we wstępnej fazie projektowania rozpocząć od ustalenia wymogów sprzętowych (rozdział 2), w oparciu o rzeczywiste możliwości użytkownika. Nieczęsto spotykane to podejście. Odnosi się zarówno do sytuacji, kiedy użytkownik dysponuje już sprzętem, jak też do sytuacji, kiedy dopiero projektuje się jego instalację. Określając kryteria doboru sprzętu, zwrócono szczególną uwagę na newralgiczne elementy systemu (jednostkę centralną, końcówki i pamięć masową) oraz podano praktyczne uwagi, jak też własne obserwacje.

Z powyższym problemem wiąże się ściśle kwestia dobrej znajomości zadań i obszaru oddziaływania bezpośredniego systemu informacyjnego (BSI) — i to nie tylko ze względu

[1]) Strześniewski Stanisław H.: Bezpośrednie systemy informacyjne. PWE, Warszawa 1985, s. 183, nakład 3000 egz. cena 96 zł

na powodzenie całego przedsięwzięcia, ale przede wszystkim na efektywne działanie systemu.

Czytelnik może zauważyć, że zdarzają się przypadki rozpoczynania procesu projektowania od szczegółowej budowy zbiorów (w książce — tematyce organizacji zbiorów w BSI poświęcony jest dopiero rozdział 5) lub technologii przetwarzania danych, co nierzadko ma niekorzystny wpływ na uwzględnienie potrzeb użytkownika.

Autor, przywiązując duże znaczenie do procesu projektowania (być może mając w pamięci własne trudności), najszerzej zaprezentował metodykę projektowania systemów o działaniu bezpośrednim (rozdział 3). Metodykę tę poszerzono o próbę naszkicowania kosztów projektowania i — co cenniejsze — wskazano jak uniknąć wielu błędów.

Książka St. H. Strześniewskiego — jak wiele innych publikacji, szczególnie tych, w których wywód teoretyczny przeplata się z treścią praktyczną — ma też pewne niedociągnięcia.

Można w niej znaleźć zbytnie uogólnienia (np. podczas omawiania współczynnika kosztów projektowania), pewne uproszczenia (np. dotyczące efektywności systemu, ochrony danych), zdarzają się przypadki nieścisłości terminologicznych, w odniesieniu do zasadniczej tematyki autor powołuje się na zaledwie kilka pozycji literatury źródłowej.

Mimo szerokiego ujęcia tematyki dotyczącej systemów o działaniu bezpośrednim, brakuje prezentacji podstawowego elementu takich systemów, a mianowicie transmisji danych. Szczególnie przydatne mogłyby tu być spostrzeżenia autora z własnej praktyki, tym cenniejsze, że literatura w tym zakresie, oparta na realiach krajowych, jest bardzo uboga. Pozostaje tylko mieć nadzieję, że zagadnienia te znajdą się w następnej publikacji autora; dopiero po tym uzupełnieniu obraz bezpośrednich systemów informacyjnych byłby pełny.

Uwagi, wskazówki i przykłady, zawarte w metodyce projektowania, podano też w pozostałych rozdziałach, m. in. dotyczących organizacji zbiorów danych i niezawodności BSI. One to przede wszystkim stanowią o wartości praktycznej książki. Jednocześnie spójność wywodu i komunikatywny język sprawiają, że publikacja ta może być adresowana do szerokiego grona Czytelników, zarówno informatyków, jak i użytkowników.

ANDRZEJ SOKOŁOWSKI
Warszawa

Dla kogo ten „podręcznik"?

Na rynku wydawniczym ukazało się niedawno wznowienie książki, która z założenia miała być podręcznikiem nauki języka ALGOL 60 i programowania w tym języku [1]). Mimo iż język ten wychodzi z użycia, jest on ciągle jeszcze wykładany w wielu szkołach wyższych. Usprawiedliwia to konieczność wydawania podręcznika tego języka dla studentów. W książce opisany jest język określony przez Autorkę jako ALGOL dla maszyn serii ODRA 1300 (czyli ALGOL 1900), a także kompilator XALM i sposób jego obsługi na ODRZE 1304, pracującej w minimalnym zestawie pod kontrolą egzekutora. Niestety, liczne błędy merytoryczne i niedociągnięcia wydawnicze nie pozwalają na pozostawienie tej książki bez komentarza.

Książka składa się z dziesięciu rozdziałów oraz aneksu, w którym podano tablice symboli ALGOLU wzorcowego i ALGOLU 1900, kody znaków na kartach i wykaz błędów sygnalizowanych przez kompilator XALM oraz nagłówki niektórych standardowych procedur wejścia i wyjścia. Brakuje w tym miejscu wykazu błędów wykonania.

Na okładce znajduje się informacja, że książka przeznaczona jest dla studentów szkół ekonomicznych oraz samouków, a we wstępie adresuje się ją do studentów II roku kierunku cybernetyki ekonomicznej. Autorka chce zapoznać czytelników z notacją Backusa-Naura, a także z zasadami pisania programów. Tymczasem formalna notacja Backusa-Naura jest mało czytelna dla początkujących i dlatego niezbyt wygodna dla dydaktyki. Ponadto notacja ta nie stosowana jest zbyt ściśle przez Autorkę (str. 60, 61, 91) oraz zawiera błędy (np. na s. 179 brak nawiasów kwadratowych w definicji zmiennej ze wskaźnikami). Wykład zaciemniają liczne błędy merytoryczne, nieprecyzyjne słownictwo i własne słowotwórstwo Autorki. Odnosi się wrażenie, że Autorka nie zna dokładnie ani działania maszyny cyfrowej, ani zasad poprawnego pisania programów. Przykłady w książce dobrane są fatalnie i nie mają żadnej wartości dydaktycznej.

W książce analizowanych jest zaledwie 8 przykładów programów. Tymczasem cytowany przez Autorkę podręcznik Paszkowskiego zawiera 30 niebanalnych przykładów, a książka Jerzykiewiczowej i Szczepkowicza o ALGOLU 1204 (w której notabene autorzy zakładają, że czytelnik zna ALGOL 60) podaje 18 przykładów programów.

Do przykładów programów też można mieć wiele zastrzeżeń. Poszczególne wersje programów minimalnie różnią się między sobą, a napisane są w sposób mało czytelny. Żaden z programów nie jest poprawnie skomentowany. W omówieniu programu pierwszego trudno było bardziej zagmatwać sposób obliczania pola i objętości walca. W dodatku zmienne robocze (pomocnicze) wprowadzono w niewłaściwy sposób. Program nr 6, obliczający sumę kwadratów 10 liczb, realizuje to rezerwując tablicę na te liczby! Nazywa się to w jego drugiej wersji „przykładem na dynamiczną rezerwację pamięci". A do tego celu pięknie nadawał się program nr 2, obliczający odchylenie przeciętne.

W dodatku w tych prostych przykładach programów, podanych jako niby wydruki z kompilacji, znajdują się błędy formalne: np. na s. 157 (tabl. 19) — występuje stała 3,14; na s. 111 (tabl. 4) — występuje instrukcja 6 : = 2*3,14159*R*(R+L): na s. 274 — w wyrażeniu pojawia się X[I] !2. Autorka przyjmuje wartość równą 3,14 (lub 3,14159), a była wspaniała okazja do zastosowania funkcji standardowej arctan dla wyznaczenia „dokładnej" wartości Π w komputerze. Złym nawykiem jest również podstawienie pod zmienną wyrażenia, którego wartość należy wydrukować, i użycie tej zmiennej zamiast całego wyrażenia w instrukcji wyjścia (nawyk z FORTRANU ?). W jednym z programów deklaruje się zmienną 0, a dalej występuje instrukcja PRINT(0,0,2). Jeśli zważyć, że ALGOL 1900 zezwala na użycie wyrażeń jako parametrów aktualnych dla wszystkich parametrów procedury PRINT, przykład ten jest szczególnie złośliwie dobrany: w którym miejscu (miejscach) jest litera O, a w których liczba 0 ?.

Wykład jest nierówny, a język nieporadny, często napuszony i sztywny. Szczegółowo wykładane są fragmenty dotyczące rzeczy banalnych i intuicyjnych, skrzętnie pomijane są natomiast zagadnienia trudne. Czytelnika denerwuje dziwna retoryka, np. wielokrotnie powtarzane są zdania takie, jak: „wyjaśnienia zostaną podane w dalszych częściach rozważań", „zagadnienie to nie jest rozpatrywane w tej książce" itp.

Nie zwracamy uwagi na fatalne „dane wejściowe", „dane wyjściowe" i „dane wynikowe" (zamiast prostszych „dane" i „wyniki"), bo tę terminologię spotyka się dosyć często, ale po co Autorka upiera się przy tym, by instrukcję nazywać zdaniem (wpływ COBOLU ?), po co wprowadza pojęcie instrukcji niebezwarunkowej (s. 140), czy zmiennych niezależnych i zależnych (s. 54)? Nie wiadomo też, co Autorka ma na myśli mówiąc o rekurencyjności zdań (s. 86 i 92). Jednocześnie sama krytykuje raport o ALGOLU 60, kwestionując istnienie programu w postaci instrukcji złożonej (s. 91—92). A przecież wiadomo, że najprostszym programem w ALGOLU jest program **begin end;** program ten można przetłumaczyć, zajmuje miejsce w pamięci, niczego nie czyta, niczego nie drukuje i ... niczego

[1]) Magdalena Szaniawska: ALGOL 60. PWE, Warszawa 1984, wyd. II, s. 292, cena 100 zł.

nie robi. Wątpliwą wartość dydaktyczną ma także przykład zapisu arkusza programowego (s. 105), w którym na początku zapisuje się co drugi wiersz (marnowanie papieru), a później trzeba wykorzystać nawet dolny margines.

W książce znajdujemy wiele nieprawdziwych stwierdzeń. Autorce mylą się kanały z urządzeniami wejścia i wyjścia, a to wszystko z numerami programowymi urządzeń. Na s. 61 Autorka twierdzi, że symbol **own** może wystąpić tylko w złożonych strukturach, na s. 185, że zmienna sterująca w instrukcji „dla" musi być zmienną prostą, a tekst na s. 167 świadczy o tym, że Autorka nie zdaje sobie sprawy z tego, że pętla „dla" może być wykonywana zero razy (występuje tu zresztą jeszcze jedno nowe pojęcie: argument instrukcji).

Są liczne powtórzenia: jak kończyć liczby-dane (s. 52 i 98), że odstępy i znaki zmiany wiersza nie są w programach istotne (s. 51, 59, 106). W wielu miejscach Autorka sama sobie przeczy: np. raz na s. 175 mówi się, że niefunkcyjna procedura OUTPUT jest typu rzeczywistego, gdzie indziej (na s. 97) mówi się o argumencie procedury funkcyjnej READ (bez parametrów).

Bardzo mało miejsca poświęca Autorka procedurom. Podane są zaledwie trzy proste przykłady tego ważnego dla programisty pojęcia językowego. Sprawa umieszczania parametrów w zbiorze wartości jest wspomniana mimochodem, a ze zdania zalecającego wywoływanie tablic przez wartość (s. 216) wynika, że Autorka nie rozumie o co tutaj chodzi.

Nie sposób podać pełnego wykazu błędów tego podręcznika: ich kompletna lista i omówienie znacznie przekraczałyby ramy recenzji i byłyby nużące dla Czytelnika.

Wydawnictwo też się nie popisało. W wielu miejscach układ tekstu nie jest uzasadniony merytorycznie i znowu marnuje się papier. W licznych miejscach pomylono zero z literą O, a nawiasy łańcuchowe są pisane jako apostrof, jako pionowa kreska (s. 146) a także jako małe nawiasy we frakcji górnej (s. 154), lub jeszcze inaczej (s. 288). Na końcu książki znajduje się spis literatury; wystarczyło go uzupełnić o inne prace cytowane w tekście lub w krótkich notkach i ponumerować je, dzięki czemu uniknięto by wielokrotnego cytowania tych samych pozycji w całej rozciągłości. Co robił redaktor, gdy czytał w maszynopisie „klawiszując z pulpitu" (s. 17 i 107)? Czy zrozumiał treść notki 8 na s. 22—23 (bo my nie)?

Odnosi się wrażenie, że nie tylko maszynopisu, ale także pierwszego wydania tej książki nikt nie czytał. Skąd bowiem tyle błędów znalazło się jeszcze w wydaniu drugim? Książka przeszła przez recenzentów i redaktorów Wydawnictwa; jak mogło dojść do tego, że książka została ponownie wydrukowana w niemałej liczbie egzemplarzy? Podobno wydawcy narzekają na brak papieru...

Nie wydaje się nam, by można było polecić tę pozycję studentom, gdyż uczenie się z tego podręcznika przynosi więcej szkody niż pożytku. Książkę można jedynie gorąco polecić informatykom znudzonym układaniem długich programów. Rozrywka zapewniona.

JERZY KUCHARCZYK
ANDRZEJ NIEMIEC
Instytut Informatyki
Uniwersytetu Wrocławskiego
Katedra Matematyki Akademii
Rolniczej we Wrocławiu

Zmienno- i stałoprzecinkowy czy zmienno- i stałopozycyjny?

Przez wiele lat na oznaczenie angielskich wyrazów **floating-point** i **fixed-point** używano w języku polskim przymiotników: zmiennoprzecinkowy i stałoprzecinkowy. Od niedawna niektóre redakcje zalecają lub zgoła wymuszają używanie w tym znaczeniu wyrazów: zmienno- i stałopozycyjny. Dręczony wątpliwościami, czy taka wymiana terminów jest uzasadniona, postanowiłem zbadać możliwie najdokładniej pochodzenie terminów angielskich i treść pojęć oznaczanych przez odpowiednie nazwy. Znaczną pomocą przy tego rodzaju przedsięwzięciu są międzynarodowe normy terminologiczne ISO. Oprócz niewątpliwie autorytatywnego charakteru mają one tę zaletę, że dopuszczają krytykę, co nie zawsze łatwo przychodzi autorytetom krajowym.

W normie ISO 2382/V wyrazy floating-point i fixed-point oznaczają pewne rodzaje systemów liczbowych, a dokładniej — systemów reprezentacji liczb. Przed podaniem i zanalizowaniem definicji tych systemów liczbowych należy jednak wprowadzić pewną systematyzację, podając kilka definicji pierwotnych.

Przez system liczbowy (ang. **numeration system**) rozumie się notację służącą do reprezentacji liczb. Z tego względu, zamiast nazwy system liczbowy często używa się nazwy pełnej — system reprezentacji liczb (ang. **number representation system**).

Warto w tym miejscu dodać, że notacją nazywa się zbiór symboli i reguł ich użycia, służących do reprezentacji danych. W określonej notacji (zwanej też po polsku zapisem) można przedstawiać nie tylko liczby, np. notacja znana pod nazwą Uniwersalnej Klasyfikacji Dziesiętnej (ang. **Universal Decimal Classification, UDC**) używa cyfr dziesiętnych do symbolizowania zawartości publikacji księgozbiorów, a tzw. Odwrotna Notacja Polska (ang. **Reserve Polish Notation, RPN**) służy do przedstawienia operacji arytmetycznych za pomocą cyfr, liter i znaków działań. Notacją dziesiętną (ang. **decimal notation**) nazywa się notację, w której używa się dziesięciu różnych znaków (ang. characters), zwykle cyfr dziesiętnych. Podobnie, notacja dwójkowa (ang. **binary notation**) jest notacją, w której używa się dwóch różnych znaków, zwykle cyfr — 0 i 1. Terminy te nie oznaczają jednak systemów reprezentacji liczb, gdyż do tego muszą być spełnione pewne dodatkowe warunki.

Istnieją różne rodzaje systemów liczbowych (systemów reprezentacji liczb). Jednym z najważniejszych jest tzw. system pozycyjny. Pozycyjnym systemem reprezentacji liczb (ang. **positional representation system, positional notation**) nazywa się dowolny system liczbowy, w którym liczba rzeczywista jest reprezentowana przez uporządkowany zbiór znaków w taki sposób, że wielkość udziału każdego znaku zależy tylko od jego pozycji i od wartości. Pozycją (ang. **digit position**) w pozycyjnym systemie reprezentacji liczb jest każde miejsce, które może być zajęte przez znak i które może być identyfikowane przez liczbę porządkową lub równoważny jej identyfikator.

Wśród systemów pozycyjnych najważniejsze są tzw. systemy liczbowe z podstawą (ang. **radix numeration systems**). System liczbowy z podstawą jest systemem pozycyjnym (tzn. pozycyjnym systemem reprezentacji liczb), w którym stosunek wagi każdej pozycji do wagi pozycji o bezpośrednio mniejszej wadze jest dodatnią liczbą całkowitą. Tę dodatnią liczbę całkowitą, przez którą mnoży

się wagę pozycji w celu otrzymania wagi pozycji o bezpośrednio większej wadze, nazywa się podstawą pozycji (ang. **radix**). Nie należy tego pojęcia mylić z tzw. bazą (ang. **base**), którą omówimy w dalszej kolejności.

Oczywiście, do pełnego wyjaśnienia definicji systemu liczbowego z podstawą należy podać, jak jest w niej rozumiany termin waga. Waga (ang. **weight, significance**) pozycji w pozycyjnym systemie reprezentacji liczb jest to czynnik, przez który mnoży się wartość reprezentowaną przez znak na tej pozycji w celu otrzymania addytywnej wielkości udziału w reprezentacji liczby rzeczywistej. Tak więc, każdej pozycji w systemie liczbowym z podstawą odpowiadają wzajemnie związane — podstawa i waga. Dopuszczalne wartości przybierane przez każdy znak w systemie z podstawą mieszczą się w zakresie od zera do liczby o jeden mniejszej od podstawy.

Wśród systemów liczbowych z podstawą wyróżnia się systemy liczbowe o stałej podstawie i systemy liczbowe o zmiennej podstawie. Systemem liczbowym o stałej podstawie (ang. **fixed radix numeration system**) nazywa się system liczbowy z podstawą, w którym wszystkie pozycje, być może z wyjątkiem pozycji o największej wadze, mają tę samą podstawę. System liczbowy o zmiennej podstawie (ang. **mixed radix numeration system**) jest to system liczbowy z podstawą, w którym poszczególne pozycje nie mają tej samej podstawy.

Można teraz podać definicje powszechnie używanych systemów liczbowych: dziesiętnego i dwójkowego systemu reprezentacji liczb. Dziesiętny system liczbowy (ang. **decimal numeration system**) jest to system o stałej podstawie, w którym używa się cyfr 0, 1, ... 9 i podstawy równej dziesięć, a najmniejsza całkowita waga równa się jeden. Czysty dwójkowy system liczbowy (ang. **pure binary numeration system**) jest to system liczbowy o stałej podstawie, w którym używa się cyfr 0 i 1 oraz podstawy równej dwa.

Sądzę, że w tym miejscu warto wyjaśnić różnicę między podstawą (ang. **radix**), której definicję już podano, a bazą (ang. **base**). Bazą systemu liczbowego nazywa się liczbę, która jest podnoszona do potęgi oznaczonej przez wykładnik i mnożona przez mantysę w celu określenia reprezentowanej liczby rzeczywistej. Systemem liczbowym o zmiennnej bazie (ang. **mixed radix numeration system**) nazywa się system liczbowy, w którym liczba jest reprezentowana jako suma ciągu składników złożonych z mantysy i bazy, przy czym stosunki baz poszczególnych składników nie muszą być całkowite.

Mówiąc obrazowo, w systemie liczbowym o zmiennej bazie określoną liczbę można przedstawić, na przykład, używając ustalonych baz, b_1, b_2, b_3 oraz określonych mantys,

m_1, m_2 i m_3, tzn.
$l = m_3b_3 + m_2b_2 + m_1b_1$

Na tym przykładzie można wyjaśnić, że system liczbowy o zmiennej podstawie jest szczególnym przypadkiem systemu liczbowego o zmiennej bazie. Zachodzi to w sytuacji, gdy stosunek baz dwóch kolejnych składników jest całkowity lecz niekoniecznie równy w każdym przypadku. Liczba l mogłaby wtedy mieć następującą reprezentację:

$l = m_3xyb + m_2xb + m_3b$

Jeżeli w systemie liczbowym o zmiennej bazie stosunek baz wszystkich par kolejnych składników jest taki sam, to system ten sprowadza się do systemu liczbowego o stałej podstawie. Liczba l może mieć wtedy następującą reprezentację:

$l = m_3x^2b + m_2xb + m_1b$

W podanych dotąd definicjach, w oryginalnym brzmieniu angielskim, nie wystąpił jeszcze żaden z terminów — **floating-point** ani **fixed-point**. Nie ma więc na razie podstaw do jakichkolwiek rozstrzygnięć w sprawie doboru ich polskich odpowiedników. Wystąpiły natomiast wyrazy: pozycja (ang. **position**), system pozycyjny (ang. **positional system**) oraz stały (ang. **fixed**) w terminie system liczbowy o stałej podstawie i zmienny (ang. **mixed**) w terminach system liczbowy o zmiennej podstawie i system liczbowy o zmiennej bazie. Wystąpienie tych wyrazów może mieć pozytywny lub negatywny wpływ na dobór odpowiedników polskich dla terminów fixed-point i floating-point. Tak więc, nazwy pozycja i pozycyjny należałoby raczej utrzymać na oznaczenie pojęć odpowiadających identycznie brzmiącym wyrazom angielskim. Podobnie, nie powinno budzić zastrzeżeń polskie tłumaczenie angielskiego wyrazu **fixed**. Wątpliwości wzbudza natomiast użycie wyrazu zmienny jako odpowiednika angielskiego **mixed**. Takie tłumaczenie (przyjęte np. w książce: S. Budkowski i in., Zespoły i urządzenia cyfrowe, WNT, Warszawa, 1979) w pełnym kontekście terminologicznym wydaje się niewłaściwe, ponieważ wyraz „zmienny" ma ugruntowane znaczenie w obu rozważanych odpowiednikach polskich terminu **floating-point**. Wydaje mi się więc, że należałoby w tym przypadku używać raczej wyrazu mieszany i mówić: system liczbowy o mieszanej podstawie i system liczbowy o mieszanej bazie.

Te, dość szczegółowe wyjaśnienia dotyczące systemów liczbowych, stanowiące tło dla właściwych rozważań na temat doboru polskich odpowiedników terminów fixed-point i floating-point, doprowadziły więc do wstępnego przyjęcia wyrazu stały na oznaczenie przedrostka **fixed** i wstępnego zastrzeżenia nazwy pozycyjny na oznaczenie angielskiego przymiotnika **positional**.

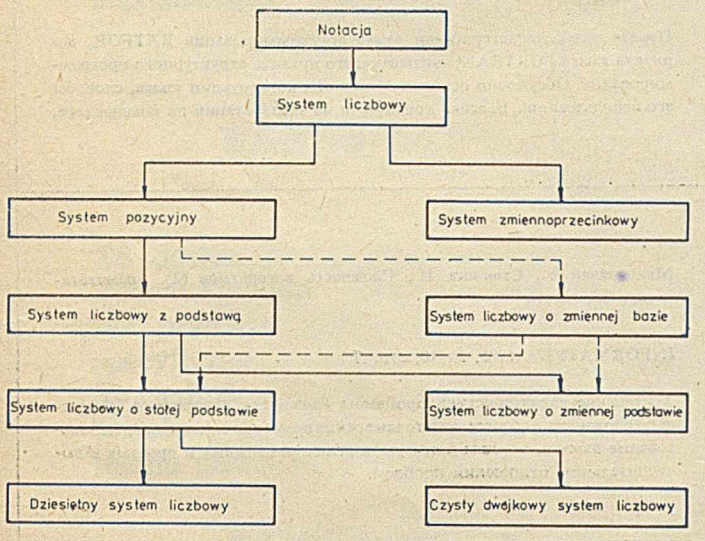
Pora teraz przystąpić do wyjaśnienia, jak tłumaczyć na język polski pozostałe słowa angielskie (tj. **point** i **floating**). W tym celu konieczne jest podanie odpowiednich definicji. Angielski termin **point**, a dokładniej **radix point**, w reprezentacji liczby wyrażonej w systemie liczbowym z podstawą, oznacza (według normy ISO 2382/V) miejsce rozdzielenia znaków związanych z częścią całkowitą liczby od znaków związanych z jej częścią ułamkową. Jest to więc, ściśle mówiąc, punkt separacji i takiego terminu polskiego używałbym jako odpowiednika terminu angielskiego **radix point**.

Można teraz podać definicje dwóch ważnych terminów angielskich: fixed-point representation system i variable-point representation system. **Fixed-point representation system**, co zgodnie z dotychczasowymi ustaleniami nazwałbym roboczo po polsku — systemem liczbowym o stałym punkcie separacji, jest to (według normy ISO 2382/V) system liczbowy z podstawą, w którym punkt separacji w ciągu pozycji jest ustalony niejawnie, zgodnie z przyjętą umową. Natomiast angielski termin **variable-point representation system**, któremu wypada nadać polską nazwę system liczbowy o zmiennym punkcie separacji (**variable** znaczy zmienny), jest to (według tej samej normy) system liczbowy z podstawą, w którym punkt separacji jest wskazany jawnie przez specjalny znak na odpowiedniej pozycji.

Oczywiście, obie nazwy systemów liczbowych — system liczbowy o stałym (zmiennym) punkcie separacji — nie spełniają kryterium zwięzłości, dlatego raczej nie będą używane. Ażeby je skrócić i wybrać najlepszy odpowiednik polski, trzeba przypomnieć, jakie znaki są używane do oznaczenia punktu separacji. Znakami tymi są, jak wszyscy wiemy, przecinek lub kropka. Przykładowo, w dziesiętnym systemie liczbowym punktem separacji jest kropka dziesiętna (przecinek dziesiętny), a w dwójkowym systemie liczbowym — kropka dwójkowa (przecinek dwójkowy). Z powyższych rozważań logicznie wynika, że angielskim terminom **fixed-point** (**variable-point**) **representation system** powinny odpowiadać polskie nazwy system liczbowy o stałym (zmiennym) położeniu przecinka (kropki), w skrócie — system stałopołożeniowoprzecinkowy i odpowiednio, zmiennopołożeniowoprzecinkowy (jeżeli chcemy zbliżyć się do nazw będących w użyciu, to w tym miejscu należy zrezygnować z uwzględnienia wyrazu kropka w ostatecznym sformułowaniu nazwy). Poddając powyższe nazwy dalszym zabiegom „odchudzającym", tzn. zachowując przedrostek i opuszczając po jednym członie zasadniczym, otrzymamy dwa alternatywne terminy — system stałopołożenio-

wy (a co za tym idzie — stałopozycyjny) i system stałoprzecinkowy (odpowiednio — zmiennopozycyjny i zmiennoprzecinkowy).

Wydaje się więc, że obie będące w użyciu nazwy polskie są pełnoprawne, gdyż w sposób skrótowy opisują dokładnie treść odpowiedniego pojęcia. Na razie jednak nie można dopuścić do ich użycia, ponieważ jedna z nich w obu wariantach, tj. system zmiennoprzecinkowy i system zmiennopozycyjny, stanowi odpowiednik angielskiego terminu **variable-point representation system**, a nie, jak chcielibyśmy, **floating-point representation system**. Należy więc wyjaśnić treść pojęcia oznaczanego tą ostatnią nazwą.



Ilustracja zależności między poszczególnymi systemami reprezentacji liczb

Termin **floating-point representation system**, któremu w dosłownym tłumaczeniu odpowiada nazwa system liczbowy o płynnym punkcie separacji, oznacza (według normy ISO 2382/V) system liczbowy, w którym liczba rzeczywista jest reprezentowana przez parę różnych liczebników w taki sposób, że jej wartość stanowi iloczyn części o stałym punkcie separacji (tj. pierwszego liczebnika czyli mantysy) i części otrzymanej przez podniesienie niejawnej bazy do potęgi oznaczonej przez wykładnik (tj. drugi liczebnik). Warto też od razu wyjaśnić, że przez liczebnik (ang. **numeral**) rozumie się dyskretną reprezentację liczby (ang. discrete representation of a number). Z zacytowanej definicji wynika jasno, że system nazwany systemem liczbowym o płynnym punkcie separacji (ang. **floating-point representation system**) trudno jest zaliczyć do systemów pozycyjnych, gdyż nie spełnia ich definicji w jej obecnym brzmieniu (p. powyżej). Obydwa liczebniki natomiast (tj. mantysa i wykładnik) są przedstawiane w systemie pozycyjnym. Zależności między omówionymi systemami liczbowymi zilustrowano na rysunku.

Jak teraz rozstrzygnąć kwestię wyboru polskiego odpowiednika dla terminu **floating-point representation system**? Postępując podobnie jak w przypadku terminów **fixed-point** i **variable-point representation system**, należałoby wybrać nazwy — system płynnoprzecinkowy lub system płynnopozycyjny. Oczywiście, nie ma wiele sensu wprowadzanie nowych nazw zamiast nazw już istniejących. Dlatego też należałoby zrezygnować z wprowadzonego w tym artykule oznaczenia angielskiego terminu **variable-point representation system** nazwą system zmiennoprzecinkowy (zmiennopozycyjny), co będzie dość bezpieczne, gdyż w praktyce tego systemu nie używa się, i zachować te nazwy na oznaczenie terminu **floating-point representation system**.

Doszliśmy więc do kwestii zasadniczej. Obie pary nazw podane w tytule tego artykułu, tj. zmienno- i stałoprzecinkowy oraz zmienno- i stałopozycyjny, są poprawne na oznaczenie angielskich terminów **floating-point** i **fixed-point**. Jednakże używanie przymiotników zmienno- i stałopozycyjny prowadzi do niepotrzebnej kolizji, ponieważ system zwany zmiennopozycyjnym (ang. **floating-point**) w zasadzie nie jest systemem pozycyjnym (ang. **positional**). Ponadto, określenie systemu pozycyjnego wiąże się z pozycją cyfry (ang. **digit position**) i od niej pochodzi jego nazwa, natomiast określenie systemu stało- i zmiennopozycyjnego — z położeniem punktu separacji (ang. **radix point**). Z kolei, używanie nazw — system zmienno- i stałoprzecinkowy — nie prowadzi do żadnych niejasności tego rodzaju, dlatego zalecałbym utrzymanie ich w użyciu zamiast dalszego rozpowszechniania nazw alternatywnych.

JANUSZ ZALEWSKI

Perycz K.: Egzekutor E6RM dla R-32

INFORMATYKA 1985, nr 10, s. 1

Charakterystyka programu umożliwiającego symulację funkcji egzekutora E6RM z oprogramowaniem komputera ODRA na komputerze R-32. Omówiono sposób realizacji oraz działania egzekutora na sprzęcie Jednolitego Systemu.

Перыч К.: Управляющая программа E6RM для Р-32

INFORMATYKA 1985. № 10, стр. 1

Характеристика программы, позволяющей симылирование функции управляющей программы Р6РМ с математического обеспечения ЭВМ ОДРА на Р-32. Обсуждено метод реализации и действия управляющей программы на технических средствах Единой Системы,

Płoski Z.: RATFOR — czyli strukturalny FORTRAN (1)

INFORMATYKA 1985, nr 10, s. 4

Pierwsza część charakterystyki języka programowania RATFOR, odmiany FORTRANU uwzględniającej zasady programowania strukturalnego. Omówiono podstawowe elementy konstrukcji języka oraz sposób jego użytkowania, ilustrując przykładami z eksploatacji na komputerze ODRA 1305.

Плоски З.: RATFOR — структурный FORTRAN (1)

INFORMATYKA 1985, № 10, стр. 4

Первая часть характеристики языка программирования RATFOR, варианта языка FORTRAN, учитывающего правила структурного программирования. Обсуждено основные элементы конструкции языка, способы его использования, поясняя примерами из эксплуатации на компьютере,

Mikołajczak B., Stokłosa J.: Złożoność algorytmów (2). Złożoność asymptotyczna

INFORMATYKA 1985, nr 10, s. 7

Dokończenie charakterystyki problemu złożoności algorytmów. Omówiono złożoność czasową i pamięciową algorytmu, ilustrując przykładami rozwiązywania problemów trudnych i łatwych pod względem obliczeniowym.

Миколайчак Б., Стоклоса И.: Сложность алгоритмов (2). Асимптотическая сложность

INFORMATYKA 1985, № 10, стр. 7

Завершение характеристики проблемы сложности алгоритмов. Обсуждено сложность алгоритмов с точки зрения времени выполнения и использования памяти, поясняя примерами решения сложных и простых в вычислительном отношении проблем.

Perycz K.: E6RM executive program for R-32

INFORMATYKA 1985, No. 10, p. 1

Characteristics of the program, that makes possible a simulation of functions of the E6RM executive program from ODRA computer software on R-32 computer. Realisation and operation methods of the executive program in environment of the Unified System hardware are discussed.

Perycz K.: E6RM-Executivprogramm für R-32

INFORMATYKA 1985, Nr. 10, S. 1

Eine Charakteristik des Programms, dass eine Simulation von Funktionen des E6RM-Exekutivprogramms aus der ODRA-Rechner Software auf R-32-Rechner ermöglicht. Es wurden Realisierungs- und Arbeitsweise des Executivprogramms auf ESER-Hardware besprochen.

Płoski Z.: RATFOR — or structural FORTRAN (1)

INFORMATYKA 1985, No. 10, p. 4

First part of characteristics of the RATFOR programming language, some variant of FORTRAN, which includes principles of structural programming. Basic elements of the language structure and its use, illustrated with examples from operating on ODRA 1305 computer, are discussed.

Płoski Z.: RATFOR — das heisst struktureller FORTRAN (1)

INFORMATYKA 1985, Nr. 10, S. 4

Erster Teil einer Charakteristik von RATFOR-Programmiersprache, einer FORTRAN-Version, die die Grundsätze der strukturellen Programmierung berücksichtigt. Es werden Grundelemente der Sprachekonstruktion und ihre Anwendungsweise, illustriert mit Beispielen von dem Betrieb auf ODRA 1305-Rechner, besprochen.

Mikołajczak B., Stokłosa J.: Algorithm complexity (2). Asympthotic complexity

INFORMATYKA 1985, No. 10, p. 7

Termination of characteristics of algorithm complexity problems. Time and store complexity of an algorithm, illustrated with examples of solving difficult and easy computing problems, are discussed.

Mikołajczak B., Stokłosa J.: Algorithmenkompliziertheit (2). Asymptotische Kompliziertheit

INFORMATYKA 1985, Nr. 10, S. 7

Beendigung einer Charakteristik von Algorithmenkompliziertheit-problemen. Es werden die zeit- und speichermässige Komplizierheit eines Algorithmus, illustiert mit Beispielen der Lösungen von schwierigen und einfachen Berechnungsproblemen, besprochen.

CSK—Computer 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

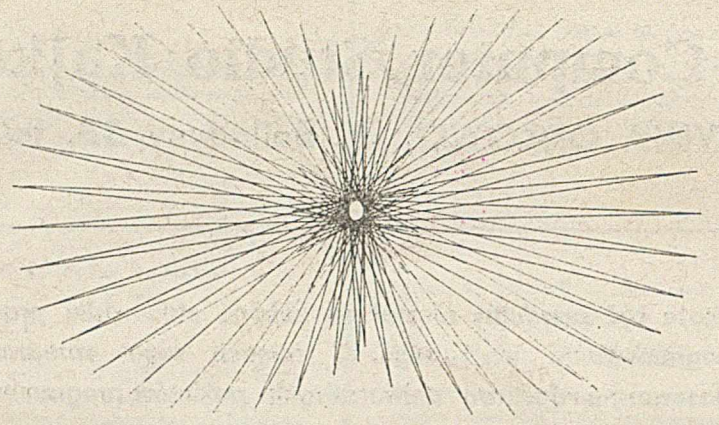
---

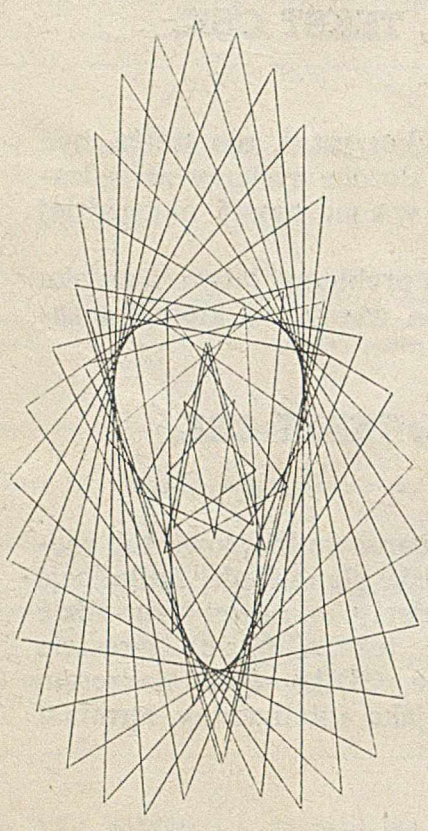
Szczegółowych informacji udziela:

CSK—Computer Studio Kajkowscy

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

EO/129/K/85





Rysunki wykonano za pomocą programu kreślącego epicykloidy przybliżane liniami łamanymi, przy czym efekt końcowy uzyskano przez:

- wykreślenie jednego rysunku lub kilku nałożonych na siebie
- różne wartości kąta obrotu układu współrzędnych i różne współczynniki skali, w kierunku osi x i osi y
- niedokładny sposób rysowania, czyli względnie dużą wartość przyrostu parametru w parametrycznych równaniach krzywych.

ZENON JĘDRZYKIEWICZ