mikroKLAN 11

11

1984

informatyka

Prekursorzy informatyki
Przerwania
Prognoza rynku

WYDAWNICTWO SIGMA

informatyka

11 1984

Nr 11
Miesięcznik Rok XIX
Listopad 1984

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. 2281. Obj. 4,0 ark. druk. Nakład 4250 egz. T-46.

INDEKS 36124

Cena egzemplarza zł 75,—
Prenumerata roczna zł 900,—

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

W NUMERZE: Strona

W NAJBLIŻSZYCH NUMERACH:

- Rafał Zieleniewski o problemach programowania mikroprocesorów
- Wiesław Wilczyński o języku PLZ
- Ryszard Rybus o programach uruchomieniowych dla mikroprocesorów
- Ryszard Kott i Krzysztof Szwed o uniwersalnych asemblerach dla mikrokomputerów
- Artur Krępski o PASCALU MT+
- Zbigniew Banasik o normie IEEE — arytmetyka zmiennoprzecinkowa
- Ryszard Kott o BASICU dla mikrokomputerów
- Waldemar Kapuścik o projektowaniu sieciowych baz danych
- Konrad Jabłoński o języku BCPL
- Wacław Iszkowski o mechanizmach komunikacji
- Marek Dziedzic, Krzysztof Perycz i Jerzy Wiliński o emulatorze E6RM-E
- Andrzej I. Litwiniuk o kompilatorze LOGLANU dla MERY 400
- Zbigniew Kierzkowski i Jacek Małuszyński o współbieżnej aktualizacji bazy danych

EMANUEL CZYŻO
TERESA LATUSEK
Uniwersytet Warszawski

Prekursorzy współczesnej informatyki

Informatyka jest bardzo młodą dziedziną wiedzy. Historia jej rozwoju przypada właściwie na okres ostatniego trzydziestolecia. Ale dawniejsze prace wielu uczonych, głównie matematyków, miały ogromny wpływ na rozwój współczesnych metod obliczeniowych. Zastosowanie ich osiągnięć przyspieszyło w znacznym stopniu automatyzację przetwarzania informacji. W związku z tym możemy ich uznać w pewnym sensie, za prekursorów współczesnej informatyki.

*

Początki swoiście pojmowanej automatyzacji rozumowania logicznego sięgają starożytnej Grecji. Związane są one głównie z imieniem wielkiego myśliciela starożytności — Arystotelesa (384—322 p.n.Ch.). Do naszych czasów zachowały się pewne utwory o charakterze literackim oraz obszerny zbiór pism filozoficznych i naukowych, wydany pod nazwę „Corpus Aristotelicum" najprawdopodobniej jego autorstwa [10].

Z naszego punktu widzenia najbardziej interesujące są osiągnięcia Arystotelesa w dziedzinie logiki. Prace jego można uznać za początki formalizacji, a zatem i automatyzacji rozumowania logicznego. On pierwszy doszedł do wniosku, że rozumowanie logiczne można opisywać i badać w sposób czysto formalny, tzn. badając formę zdań, a pomijając sens wchodzących w te zdania słów. Oddzieliwszy logikę od filozofii, jako narzędzie metodologiczne wszystkich nauk, stworzył on pierwszy fragment logiki formalnej. Ukazał jej zagadnienia i metody, wyróżnił zasadę sprzeczności oraz zasadę wyłączonego środka. Zbudował teorię wnioskowania bezpośredniego, sylogistykę zdań kategorycznych i teorię definicji. Wprowadził również podstawową terminologię logiki. Pierwszy zastosował zmienne nazwowe i metodę formalnego dowodzenia. Poddał analizie technikę prowadzenia dyskusji i dał pierwszy przegląd błędów argumentacji. Sylogistyka Arystotelesa bez zasadniczych zmian przetrwała do końca XIX wieku i stała się częścią składową współczesnej logiki formalnej [18].

*

Centralną postacią grupy matematyków i astronomów Arabii IX wieku był Abu Jafar Muhammad ibn Musa al-Khwarizmi (Alchwarizmi). Swoje miejsce w historii matematyki zawdzięcza on głównie dwom rozprawom matematycznym. Arytmetyczny traktat Alchwarizmiego — „O liczbach i działaniach na nich", był pierwszym arabskim dziełem, w którym wyłożony został dziesiętny system pozycyjny i oparte na nim działania arytmetyczne. Co prawda dziesiętny system pozycyjny znany był już od dawna (IV w.n.e.) Hindusom; posługiwano się w nim dziewięcioma cyframi i pustym miejscem, które dopiero znacznie później zaczęto oznaczać zerem [8]. Jednak dopiero rozpropagowanie tego systemu przez Arabów, sprawiło, że stał się on podstawowym systemem rachowania używanym powszechnie na całym świecie. Nic więc dziwnego, że cyfry 0, 1, 2,...,9 nazywane są cyframi arabskimi. Przyznać też trzeba, że stało się tak pomimo tego, że początkowo mówiło się wyraźnie o „nowej indyjskiej sztuce rachowania" i nawet w tłumaczeniu łacińskim dzieł Alchwarizmiego (w. XIII) używano jeszcze terminu — de numero Indorium.

W drugim traktacie zatytułowanym — „Kitab al-djabr w al-mukabala", Alchwarizmi podaje arytmetyczne i geometryczne metody rozwiązywania równań pierwszego i drugiego stopnia wyprowadzając uzupełnienia (likwidacja wyrazów ujemnych przez dodanie do obu stron wyrazów przeciwnych) i wyrównanie (redukcja wyrazów podobnych [7].

Zachowało się, w formie częściowo zmodyfikowanej, pięć jego dzieł: o arytmetyce, algebrze, astronomii, geografii i kalendarzu. Wywarły one ogromny wpływ na dalszy rozwój matematyki, a wiele ich fragmentów weszło do innych prac. Uczyły się na nich dziesiątki pokoleń, bowiem Alchwarizmi zebrał w nich wszystko, co miało wówczas wartość zarówno dla uczonych, jak i dla praktyków. Przeprowadzając obliczenia w systemie dziesiętnym opracował tablice astronomiczne. Od jego nazwiska wywodzi się termin „algorytm", a z tytułu drugiego traktatu wziął swój początek termin „algebra".

*

W 1550 roku urodził się John Napier (Jan Neper), baron szkocki znany z odkrycia logarytmów. Jego dziełem jest urządzenie (zbudowane ze specjalnych sztabek), zwane „laską Nepera" lub „kostkami Nepera", służące do wykonywania łatwego mnożenia i dzielenia na liczbach wielocyfrowych bez znajomości tabliczki mnożenia. Pomysł swych liczydeł oparł Napier na hinduskim sposobie mnożenia, rozpowszechnionym w Europie pod nazwą „gelosia" (czyli zazdrość — od żaluzji, które przypomina swym kształtem krata używana w tym mnożeniu, zwanych „zazdrostkami"). Przyrząd ten, mimo swej prostoty i taniości, nie znalazł jednak większego zastosowania [6].

Bardzo wygodna i szeroko obecnie stosowana notacja ułamków za pomocą kropki dziesiętnej, to także jego pomysł.

Napier ułożył pierwsze tablice logarytmiczne — „Mirifici logarithmorum canonis descriptio". Tablice logarytmiczne Napiera nie były ułożone zbyt zręcznie, jednak — modyfikowane przez innych uczonych — ułatwiały w znacznym stopniu przeprowadzanie żmudnych rachunków w żywiołowo rozwijającej się astronomii [7].

*

Wilhelm Schickard (1592—1635), profesor języków biblijnych, matematyki, astronomii i geodezji w Tybindze, wynalazł w 1623 roku pierwszą maszynę do dodawania i odejmowania, w której było wykonywane przeniesienie cyfr dziesiętnych przez sześć miejsc. Wykonywała ona również mnożenie i dzielenie za pomocą tabliczki mnożenia. Maszyna ta zapewne nieźle działała, skoro jeden z konstruowanych egzemplarzy przeznaczony był dla Keplera [8].

*

Jednak pierwszą prawdziwą maszynę do liczenia wynalazł Francuz Błażej Pascal [16]. Bardzo wcześnie zainteresował się on problemami matematycznymi i już w 1639 roku (mając 16 lat) napisał traktat o przecięciach stożkowych. W 1642 roku zaprezentował skonstruowaną własnoręcznie maszynę do sumowania, opartą na zasadzie obrotowych kół zębatych. Liczby były przedstawione za pomocą wielkości kątowych. Z rozwiązań przyjętych przez Pascala na podkreślenie zasługuje możliwość automatycznego przeniesienia dziesiątkowego i wykonywanie odejmowania przez odwrotny obrót tarczy liczącej. Natomiast mnożenie sprowadzało się jeszcze do wielokrotnego dodawania. Maszyna ta była maszyną jednookresową, tzn. nastawianie cyfry i otrzymywanie wyniku realizowane było w tym samym cyklu pracy. Przy użyciu tego sumatora, dodawanie dwóch liczb sprowadzało się do sumowania cyfry z cyfrą. Z tej przyczyny nazywany jest on sumatorem szeregowym [15]. Pascal uznawany jest za współtwórcę (wraz z Piotrem Fermatem) matematycznej teorii prawdopodobieństwa, zapoczątkowanej rozwiązywaniem problemów dotyczących gier hazardowych. Jego traktat o znanym „trójkącie arytmetycznym" utworzonym przez współczynniki dwumianowe został ogłoszony już po jego śmierci, w roku 1665.

*

Maszynę skonstruowaną przez Pascala udoskonalił w znacznym stopniu niemiecki uczony Gotfryd Wilhelm Leibniz [8]. Pascal zmechanizował proces dodawania, natomiast Leibniz uczynił to z procesem mnożenia. Opracował on (ok. 1647 roku) model arytmometru zwanego sumatorem równoległym. Konstrukcja modelu przewidywała wykonywanie czterech działań arytmetycznych. Arytmometr był maszyną dwuokresową, tzn. w jednym cyklu nastawiano liczbę, natomiast w drugim — za pomocą ruchu korbką — nastawione cyfry były wprowadzane do licznika.

Leibniz skonstruował taki asortyment w 1694 roku. Maszyna ta miała ruchomą karetkę i ustawiony na stałe mnożnik. Podstawową rolę spełniało w niej urządzenie składające się z dwóch kół zębatych. Większe, przypominające schodkowy walec, zawierało dziewięć „zębów-schodków" różnej długości. Długość najdłuższego „zęba" odpowiadała cyfrze 9, długość następnego — cyfrze 8, itd. Mniejsze koło zębate, mające dziewięć zębów, w zależności od pozycji zaczepiało o odpowiedni ząb na walcu. Te dwa koła umieszczone były na równoległych oskach, przy czym mniejsze koło mogło być przesuwane wzdłuż oski.

Leibniz zamierzał również skonstruować maszynę, za pomocą której można by było obliczać pierwiastki, ale pomysłu tego nie zrealizował. W XVII wieku zarówno arytmometr Leibniza, jak i udoskonalone przez jego następców maszyny traktowano jako swego rodzaju ciekawostki, nie używano ich do celów praktycznych. Powodem tego były zarówno wysokie koszty wytwarzania, jak i spora jeszcze zawodność tych urządzeń.

Leibnizowi zawdzięczamy także ogłoszenie zasad rachunku różniczkowego i całkowego, opracowanie reguły całkowania funkcji oraz podanie metody przybliżonego całkowania graficznego.

Wielkim zamierzeniem Leibniza było stworzenie uniwersalnego języka symbolicznego, początkowo do opisywania logiki, a następnie — całej filozofii. Zamysł ten zrodził się zapewne z wnikliwej analizy ówczesnej matematyki, gdzie zdaniem Leibniza rozumowania przybrały postać rachunków, w sensie przekształcania napisów, operowania symbolami. A ponieważ języki naturalne nie mogą być tutaj użyte ze względu na występujące w nich wieloznaczności, należy zatem stworzyć specjalny język formalny, umożliwiający precyzyjne rozumowania — tak, aby w przypadku powstania wśród filozofów różnicy zdań można było ów spór wyrazić w tym języku i — po przeprowadzeniu odpowiednich rachunków — rozstrzygnąć o prawdzie. Leibniz zapewniał, że z pomocą kilku „inteligentnych ludzi" jest w stanie zbudować taki rachunek w ciągu paru lat.

Spośród wielu jego pomysłów na uwagę zasługuje jeszcze system dwójkowy, którego był twórcą i w którym przeprowadzał wiele rachunków, używając tylko symbolu „nic", czyli 0, i cyfry 1 — dla zapisania każdej liczby.

Skonstruowanie machiny liczącej, dążenie do stworzenia formalnego języka dla opisu istniejącego stanu wiedzy, a także inne pomysły dotyczące automatyzacji, w tym np. przekonanie, iż „nie ulega najmniejszej wątpliwości, że jakiś człowiek mógłby skonstruować maszynę, która byłaby zdolna przechadzać się jakiś czas po mieście i skręcać właśnie na rogach ulic" [5], w swoim czasie pozwoliły N. Wienerowi uznać Leibniza patronem cybernetyki.

*

W 1728 roku Falckon wynalazł warsztat tkacki, w którym posługiwano się kartami dziurkowanymi w postaci drewnianej płyty z systemem wywierconych otworów. A już w kilkadziesiąt lat później (1801) Joseph Jacquard z Lyonu wynalazł automatyczny warsztat tkacki, w którym zastosował zasadę przedstawiania informacji przez rozmieszczenie otworów w karcie papierowej. Za pomocą tych otworów sterowano zmianą kolorów przędzy przy produkcji wielobarwnej tkaniny. Powszechnie uważa się, że zastosowana tu karta papierowa stała się pierwowzorem używanych potem powszechnie kart dziurkowanych [8].

*

Do grona prekursorów informatyki włączymy także polskiego konstruktora, pochodzenia żydowskiego — Abrahama Sterna z Hrubieszowa [20]. Już od najmłodszych lat interesował się mechaniką, rodzice oddali go więc do terminu u jedynego w tym miasteczku zegarmistrza, gdyż na edukację szkolną nie mieli pieniędzy. Uzdolnienia i zmysł konstruktorski dość szybko przyniosły mu popularność w okolicy. Pracami młodego mistrza zainteresował się ks. Stanisław Staszic, podczas swej bytności w Hrubieszowie. Odtąd Stern rozwijał swój talent pod opiekuńczym okiem ks. Staszica. Mając trzydzieści lat pracowity mechanik przeniósł się wraz z rodziną do Warszawy, gdzie od podstaw rozpoczął na własną rękę edukację, nadrabiając minione lata.

W styczniu 1813 Stern zreferował, na posiedzeniu Towarzystwa Przyjaciół Nauk, zasady działania skonstruowanej przez siebie maszyny cyfrowej, wykonującej cztery podstawowe działania arytmetyczne. Była to prosta w użyciu maszyna wykonana jednak z nietrwałych materiałów, skutkiem czego szybko się psuła, a biedny wynalazca nie miał funduszy na solidniejszą konstrukcję. Wciąż jednak, w miarę możliwości, pracował nad jej ulepszeniem. Jego maszyna umożliwiała także wyciąganie pierwiastków oraz obliczanie ułamków, lecz w praktyce nie znalazła zastosowania. Karierę zrobiła dopiero w 1874 roku, kiedy to szwedzki inżynier W. Odhner, zatrudniony w Wytwórni Papierów Wartościowych w Petersburgu, opracował projekt maszyny wzorowanej na maszynie Sterna. Typ tej maszyny przyjął się i podjęto seryjną produkcję. Na zasadach tej konstrukcji oparta jest zresztą budowa używanych do dnia dzisiejszego arytmometrów.

*

Mniej więcej w tym samym czasie działał znakomity matematyk angielski Charles Babbage (1790—1871). Wkrótce po skończeniu studiów, dzięki licznym rozprawom naukowym, włączony został — mimo dość młodego wieku — do grona angielskich uczonych. Zajmował się geometrią, magnetyzmem i teorią gier hazardowych, ale najbardziej pochłaniały go prace konstrukcyjne [11].

W 1812 roku opracował koncepcję „machiny różnicowej", przeznaczonej do obliczania i drukowania tablic matematycznych. Budowa tej machiny została ukończona po dziesięciu latach. Zaś w 1833 roku Babbage zaprojektował „maszynę analityczną" — o zasadzie działania zbliżonej do współczesnych maszyn cyfrowych. U jej podstaw leżała idea programowego sterowania przebiegiem obliczeń. Miała ona wykonywać podstawowe działania arytmetyczne, zapamiętywać dane wejściowe oraz pośrednie i końcowe wyniki obliczeń, a także program sterujący przebiegiem obliczeń. Na szczególne podkreślenie zosługuje wprowadzenie programu i danych z kart dziurkowanych (według zasady Jacquarda) oraz możliwość modyfikacji obliczeń w zależności od uzyskanych wyników. Istniejący wówczas poziom techniki nie pozwolił jednak na zrealizowanie tej oryginalnej idei Babbage'a [2].

*

W latach pięćdziesiątych XIX wieku angielski matematyk i logik George Boole stworzył dwuelementową algebrę logiki. W ówczesnych czasach jego teoria nie znalazła żadnego zastosowania i nawet sam autor, zafascynowany prostotą tej logiki, nie był w stanie ocenić znaczenia własnego wynalazku. Obecnie algebry Boole'a są podstawą logiki matematycznej, teorii sieci telekomunikacyjnych i teorii elektronicznych maszyn cyfrowych.

*

W roku 1876 bracia Thomsonowie publikują opis mechanicznej metody całkowania, co uważane jest za początek rozwoju nowoczesnych maszyn analogowych. Prym w pracach naukowych wiódł starszy z braci, Wiliam Thomson (1824—1907) — angielski fizyk i matematyk, który za zasługi naukowe otrzymał tytuł lorda Kelvina. Był on jednym z najbardziej znanych badaczy przyrody w XIX wieku. Prowadził prace w wielu dziedzinach fizyki, techniki i matematyki. Skonstruował również wiele przyrządów, m.in. dla informatyki — mechaniczny analizator równań różniczkowych [9].

*

Amerykański konstruktor — Herman Hollerith, żyjący w latach 1860—1929, opracował mechaniczny system rejestrowania danych przez wycinanie dziurek w karcie papiero-

wej. Zbudował także maszynę do sortowania i zliczania opracowanych danych, przedstawionych za pomocą takich właśnie kart.

Dziurkowany karton papierowy — stanowiący pierwowzór obecnie stosowanych kart — przesuwany był ręcznie. Rząd igieł teleskopowych badał kartę; w przypadku stwierdzenia dziurki zamykał obwód elektryczny, powodując w ten sposób zwiększenie stanu odpowiedniego licznika o 1.

Hollerith był w owym czasie pracownikiem urzędu statystycznego USA i opracował tę maszynę na potrzeby spisu ludności. Do tego celu wykorzystywana była jeszcze dwukrotnie (1891 — Kanada i 1897 — Rosja). Zapoczątkowało to rozwój maszyn licząco-analitycznych o elektronicznym odczycie danych, pracujących w oparciu o karty dziurkowane. Po dziewięciu latach od momentu opatentowania wynalazku, Hollerith zorganizował przedsiębiorstwo pod nazwą „Tabulating Machine Company", produkujące i sprzedające maszyny jego konstrukcji. W roku 1924, po znacznym rozszerzeniu działalności, przekształcono to przedsiębiorstwo w firmę IBM [3].

*

Sztuka liczenia, którą początkowo zajmowali się głównie matematycy, traciła stopniowo swoje znaczenie, w miarę jej opanowywania przez coraz szersze kręgi rachmistrzów. Zagadnienia obliczeniowe stały się ponownie jednym z ważniejszych zagadnień matematycznych dopiero w początkach XX wieku. Dla wielu interesujących problemów znane były metody ich rozwiązania w sposób mechaniczny (algorytmiczny). Ale pojawiły się także problemy, co do których nie było już pewności, czy można je rozwiązać w sposób algorytmiczny. Aby pokazać, że dany problem można rozwiązać algorytmicznie, wystarczy podać odpowiedni sposób postępowania — konkretny algorytm. Do tego celu w zupełności wystarcza intuicyjne pojęcie algorytmu, które znane było już od wieków. Ale jak wykazać (udowodnić), że dany problem nie da się rozwiązać w sposób algorytmiczny? Stąd zrodziła się potrzeba zdefiniowania pojęcia algorytmu w sposób ścisły na gruncie matematyki. Sztuki tej, tzn. formalnego określenia pojęcia algorytmu, dokonali między innymi: A. M. Turing, E. L. Post, S. C. Kleene i A. Church. Najbardziej użyteczne na potrzeby informatyki okazało się podejście Turinga i Posta, którzy określili pojęcie algorytmu poprzez działanie odpowiedniej maszyny, zwanej obecnie maszyną Turinga [1].

*

Do grona nawybitniejszych matematyków naszego stulecia należy niewątpliwie matematyk amerykański, pochodzenia węgierskiego, John von Neumann (1903—1957). W początkowym okresie swej działalności naukowej zajmował się głównie zagadnieniami teoretycznymi, wnosząc istotny wkład w rozwój wielu działów matematyki i fizyki. Zadziwiająca jest różnorodność jego zainteresowań [14, 19]. Pod koniec lat dwudziestych zajmował się logiką symboliczną, teorią zbiorów i teorią dowodzenia. Interesował się także i wniósł wielki wkład w rozwój matematycznych podstaw teorii kwantów, teorii ergodyczności i teorii operatorów. Z roku 1928 pochodzi jego słynny artykuł poświęcony teorii gier. W początkach lat trzydziestych pracował nad teorią struktur, geometrią różniczkową i algebrami Boole'a. Natomiast w końcu tej dekady zainteresował się hydrodynamiką, dynamiką i mechaniką ośrodków ciągłych w odniesieniu do techniki jądrowej i meteorologii. W okresie II wojny światowej „zniknął na Zachodzie", pracując na potrzeby armii USA [12].

Badania dotyczące hydrodynamiki, a zwłaszcza ogromne trudności w rozwiązywaniu równań różniczkowych cząstkowych metodami analitycznymi, skierowały zainteresowania von Neumanna na metody przybliżone i na wykorzystanie szybkich maszyn liczących — jako narzędzia w pracach teoretycznych. Główne wyniki w zakresie metod numerycznych dotyczą metod odwracania macierzy, analizy błędów zaokrągleń i metod Monte Carlo [4, 13, 14]. Z roku 1944 pochodzi jego słynna książka, napisana wspólnie z O. Morgensternem — „Theory of games and economic behavior", która (wraz z artykułem z 1928 roku) daje początek współczesnej teorii gier [19].

W latach 1943—1946 w Uniwersytecie Pensylwanii powstaje Electronic Numerical Integretor and Computer (ENIAC) opracowany przez J. G. Brainerda, A. W. Burksa, H. H. Goldstinea, A. Goldstinea, T. K. Sharplessa i grupę inżynierów. John von Neumann zainteresował się tym projektem już w trakcie jego realizacji i niebawem został powołany na konsultanta. Analizując projekt doszedł do wielu odmiennych rozwiązań, bądź modyfikacji, tak że w efekcie powstała nowa koncepcja maszyny cyfrowej, różniąca się istotnie od ENIACA. Nowy projekt, opracowany wspólnie przez cały zespół, otrzymał nazwę — Electronic Discrete Variable Automatic Computer i został ostatecznie zrealizowany w Uniwersytecie Pensylwanii.

W tym czasie J. von Neumann, A. W. Burks i H. H. Goldstine pracowali już nad projektem nowej maszyny cyfrowej. Maszyna ta, którą na cześć von Neumanna nazwano: JONIAC, została zrealizowana w Princeton w Institute for Advanced Study. Idee przyjęte zarówno w projekcie EDVAC, jak i w komputerze JONIC — dotyczące: wprowadzenia systemu binarnego w miejsce systemu dziesiętnego użytego w maszynie ENIAC, organizacji poszczególnych elementów maszyny, w szczególności pamięci, oraz organizacji i funkcjonowania komputera jako całości — stały się podstawą wszystkich konstruowanych później komputerów.

Zasługą von Neumanna było opracowanie nowego podejścia do tworzenia maszyny cyfrowej, oddzielającego projektowanie od konstruowania komputera. Pracował też nad usprawnieniem programowania, wprowadzając do opisywania algorytmów schematy blokowe a także zajmując się biblioteką podprogramów oraz programem tworzącym program właściwy z programów znajdujących się w bibiliotece podprogramów. Mówił też o języku programisty, w odróżnieniu od języka wewnętrznego maszyny, i o translatorze z tego języka na język maszyny, posługując się terminami kod krótki (język programisty) i kod zupełny (język wewnętrzny maszyny) [12]. Sam opracowywał algorytmy i pisał programy. W dowód uznania doniosłego wkładu w rozwój maszyn cyfrowych i metod ich wykorzystania, von Neumann otrzymał medal Enrico Fermiego; uznawany jest powszechnie za ojca współczesnych maszyn cyfrowych.

W powiązaniu z tą tematyką, von Neuman prowadził badania dotyczące funkcjonowania mózgu ludzkiego [12] i różnego rodzaju automatów [17]. Pod koniec lat czterdziestych zaczął tworzyć teorię automatów, rozumianą nieco odmiennie od innych (np. od Wienera), jako ogólną teorię obejmującą swym zasięgiem maszyny analogowe i maszyny cyfrowe, a także automaty sztuczne i automaty rzeczywiste, obejmujące również organizmy żywe [13].

Niestety, będąc w pełni sił twórczych i mając przed sobą wiele projektów badań i wiele niedokończonych prac, zaczął nagle opadać z sił. Zmarł, po dwóch latach cierpień, w 1957 roku.

Wszystkie jego prace wydano w zbiorze [14]. Ostatnie niedokończone wykłady zostały opracowane przez A. W. Burksa i opublikowane w pracy [13]. Obszerny opis największych osiągnięć Johna von Neumanna można znaleźć w 64 tomie (No. 3, część 2, 1958) Biuletynu Amerykańskiego Towarzystwa Matematycznego.

*

Z dorobku naukowego i osiągnięć przedstawionych tutaj uczonych i konstruktorów korzysta obecnie olbrzymia rzesza ludzi związanych z informatyką, z jej rozwojem i zastosowaniami. Oczywiście, lista uczonych, których bez zastrzeżeń można by uznać za prekursorów informatyki, jest znacznie bogatsza. Wybraliśmy tutaj tylko niektórych z nich. Należałoby również więcej miejsca poświęcić każdej z prezentowanych tutaj postaci, ale — jak widać — temat jest bardzo obszerny i trudno przedstawić go bez koniecznych ograniczeń. Dotychczas brak jest bardziej systematycznych badań i opracowań poświęconych „korzeniom" informatyki. Mamy więc nadzieję, że ten szkic przyczyni się do większego zainteresowania tą ciekawą i ważną tematyką, stanowiącą element składowy ogólnej kultury informatycznej.

LITERATURA

[1] Davis M.: Computability and unsolvability. Mc Grawth Book Company, New York, 1958

[2] Desmonde W. H.: Maszyny matematyczne i ich zastosowanie. PWN, Warszawa, 1969

[3] Empacher A.: Maszyny liczą same? Wiedza Powszechna, Warszawa, 1960

[4] Goldstine H. H., von Neumann J.: Numerical inverting of matrices of high order. Bull. Amer. Math. Soc., vol. 53, 1947, pp. 1021—1099
[5] Gordon M.: Leibniz. Wiedza Powszechna, Warszawa, 1974
[6] Jeleński S.: Śladami Pitagorasa. PZWSz, Warszawa, 1953
[7] Juszkiewicz A. P.: Historia matematyki w wiekach średnich. PWN, Warszawa, 1969
[8] Kaufmann H.: Dzieje komputerów. PWN, Warszawa, 1980
[9] von Lane M.: Historia fizyki. PWN, Kraków, 1960
[10] Leśniak K.: Arystoteles. Wiedza Powszechna, Warszawa, 1965
[11] Moseley M.: Irascible genius — a life of Charles Babbage, inventor, Hutchinson of London, 1964
[12] Von Neumann J.: Maszyna matematyczna i mózg ludzki. PWN, Warszawa, 1963
[13] von Neumann J.: Theory of self-reproducing automata. Ed. Burks A. W., University of Illinois Press, 1966
[14] von Neumann J.: Collected works. vol. 6, New York, 1961—1963 (tom 5 — Design of computers, theory of automata and numerical analysis)
[15] Pascal B.: Rozprawy i listy, PAX, Warszawa, 1962
[16] Płużyński T.: Pascal. Wiedza Powszechna, Warszawa, 1974
[17] Shannon C. E.: Von Neumann's contributions to automata theory. Bull. Amer. Math. Soc., vol. 64, No. 3, Part 2, 1958, pp. 123—129
[18] Tatarkiewicz W.: Historia filozofii. PWN, Warszawa, 1978
[19] Ulam S. M.: John von Neumann, 1903—1957. Bull. Amer. Math. Soc. vol. 64, No. 3, Part 2, 1958, pp. 1—49
[20] Winnicka H.: Zapomniany wynalazca. PZWSz, Warszawa, 1962.

UWE PETERMANN
Sektion Mathematik
Karl-Marx — Universitaet Leipzig

ANDRZEJ SZAŁAS
Instytut Informatyki
Uniwersytet Warszawski

Przerwania — alternatywny mechanizm komunikacji między procesami współbieżnymi

W końcu lat sześćdziesiątych i w pierwszej połowie lat siedemdziesiątych panującym modelem obliczeń współbieżnych były procesy interakcyjne (ze wspólnymi obszarami danych). Wzrastające zainteresowanie techniką mikroprocesorową oraz rozwój sieci komputerowych spowodowały, iż ten wygodny z programistycznego punktu widzenia model został wyparty przez bardziej realistyczne, lepiej odpowiadające sprzętowej rzeczywistości procesy rozproszone (bez wspólnej pamięci).

Za punkt przełomowy umownie przyjąć tu można opublikowaną w 1978 roku pracę [4], w której C.A.R. Hoare przedstawił propozycję języka programowania CSP (Communicating Sequential Processes). O ile przesyłanie informacji pomiędzy procesami interakcyjnymi odbywało się zazwyczaj za pośrednictwem zmiennych dzielonych (występujących najczęściej w rejonach krytycznych lub monitorach — por. [2, 5]), o tyle w nowym modelu niezbędne okazały się odmienne konstrukcje językowe — umożliwiające komunikację procesów. W CSP jako mechanizm komunikacji przyjęto tzw. instrukcje wejścia-wyjścia. W tym przypadku „dialog" między procesami możliwy jest jedynie wówczas, gdy proces-nadawca usiłuje wysłać komunikat, zaś proces-adresat chce ten komunikat odebrać. W podobnym kierunku poszli twórcy ADY [10], wprowadzając mechanizm spotkania (ang. rendez-vous).

W CSP i ADZIE (oraz w wielu innych językach stosujących odmienne od spotkań metody komunikacji, takich jak AMPL [3] czy CHILL [6]) ważnym ograniczeniem jest niemożność przeprowadzenia komunikacji między procesami, o ile adresat nie wykona instrukcji odbioru komunikatu.

Inna koncepcja komunikacji procesów, stosunkowo najbliższa prezentowanej przez nas, zaproponowana została w 1978 roku przez P. Brinch Hansena [1]. Procesy przesyłają tu informacje za pośrednictwem procedur, których działanie wzajemnie się wyklucza. Poważnym mankamentem języka Brich Hansena jest niemożność dynamicznego tworzenia i usuwania procesów oraz przerywania działających procedur przez inne, których wykonanie jest — być może — znacznie pilniejsze.

Celem poniższego artykułu (opartego głównie na raporcie [9]) jest przedstawienie takiej metody przesyłania informacji między procesami, która nie stwarza wymienionych wyżej ograniczeń. Komunikacja odbywa się w tym przypadku za pomocą nadawania i obsługi sygnałów — przerwań (ang. interrupts). Proponowane rozwiązanie stanowi rozszerzenie mechanizmu przerwań występującego w większości współczesnych komputerów na poziomie języka wewnętrznego. Podstawowe trudności we wprowadzeniu przerwań do języków wysokiego poziomu polegają na zapewnieniu niesprzeczności z innymi konstrukcjami językowymi oraz zachodzenia niezmienników „running-systemu".

Pierwszy z tych problemów został rozwiązany w przedstawionym poniżej prostym języku PCI (Processes Communicationg by Interrupts) oraz — w przypadku procesów interakcyjnych — w znacznie bardziej skomplikowanym LOGLANIE 84 [7].

Pokonanie drugiej trudności wymaga narzucenia na generator kodu kompilatora następujących ograniczeń:

- proces może być przerwany tylko w takich momentach, w których stan jego obliczeń odpowiada stanowi jego pamięci
- miejsca, w których może nastąpić przerwanie, powinny się pojawiać dostatecznie często (zależnie od potrzeb programisty).

Narzędzie, jakim w prezentowanym ujęciu są przerwania, jest przydatne w jednolitym tworzeniu wszystkich warstw oprogramowania: od współpracy systemu operacyjnego ze sprzętem maszyny począwszy, a skończywszy na programach użytkowych (por. [3, 11]). Szczególnie przekonywającymi obszarami zastosowań są naszym zdaniem systemy, w których procesy muszą szybko reagować na zdarzenia (żądania) zgłaszane w sposób asynchroniczny. Do zastosowań tych zaliczyć można m.in.:
— tworzenie systemów czasu rzeczywistego do sterowania przebiegiem procesów przemysłowych i eksperymentów naukowych
— programowanie rozproszonych baz danych
— tworzenie oprogramowania sieci komputerowych.

Ponadto za pomocą przerwań można definiować i efektywnie programować inne mechanizmy komunikacji (np. wspomniane wcześniej rozwiązania występujące w CSP, ADZIE, AMPL, CHILLU czy języku Brinch Hansena) oraz rozmaite narzędzia synchronizacji.

Ze względu na ograniczoną objętość artykułu zdecydowaliśmy się przedstawić w nim „czysty" mechanizm przerwań. Opisany bardzo prosty język można rozszerzyć o ta-

blice, procedury, funkcje i współprogramy. Doświadczenie z wprowadzeniem przerwań do LOGLANU 84 wykazuje możliwość ich zastosowania także w znacznie bardziej wyrafinowanych językach programowania. W tym ostatnim przypadku przerwania stwarzają dodatkowo możliwości programowania obsługi sytuacji wyjątkowych (ang. exception handling) występujących w obliczeniach współbieżnych (por. [12]).

SKŁADNIA I REGUŁY WIDOCZNOŚCI

Do opisu składni języka PCI użyjemy rozszerzonej notacji BNF, w której:
— symbole nieterminalne zawarte są w nawiasach <>
— symbolami terminalnymi są & ; : = . , : oraz słowa wytłuszczone
— napisy opcjonalne umieszczone są w nawiasach kwadratowych [ ]
— nawiasy sześcienne { } służą do oznaczenia iteracji.

Oto najważniejsze reguły gramatyki:

<system> : : = <deklaracja procesu>
<deklaracje> : : = {<deklaracja procesu> | <deklaracja zmiennych> |
<deklaracja modułu obsługi przerwania>}
<deklaracja procesu> : : =
process <identyfikator> [<lista parametrów formalnych>];
<deklaracje>
[begin <instrukcje> end] [<identyfikator>] ;
<deklaracja modułu obsługi przerwania> : : =
handler <identyfikator> [<lista parametrów formalnych>] ;
{<deklaracja zmiennych>}
[begin <instrukcje>] end [<identyfikator>] ;
<deklaracja zmiennych> : : =
var <identyfikator> {, <identyfikator>} : <typ>
{, <identyfikator> {, <identyfikator>} : <typ> ;
<typ> : : = boolean | character | integer | process | real |
<identyfikator typu procesu>

<instrukcje> : : = [<instrukcja>] {<separator> <instrukcja>}
<separator> : : = ; | &
<instrukcja> : : =
<idvp> : = start <idp> [<lista parametrów aktualnych>] |
activate | passivate | terminate | mask [<idh>] {, <idh> } |

mask all | return | <podstawienie> | <złożenie> | <selekcja> |
<iteracja> |
interrupt <idvp> . <idh> [<lista parametrów aktualnych> [answer <idvb>]

Ze względu na pobieżny charakter prezentacji, znaczenie niektórych symboli nieterminalnych pozostawiliśmy domyślności czytelników.

Zwróćmy jeszcze uwagę na ważniejsze warunki kontekstowe:
— <idvp> oznacza identyfikator zmiennej referencyjnej; <idvb> — identyfikator zmiennej logicznej; <idh> — identyfikator modułu obsługi przerwania (handlera)
— <idp> oznacza identyfikator typu procesu; w instrukcji <idvp> : = **start** <idp> ... zmienna <idvp> musi być albo typu <idp>, albo typu **process**
— lista parametrów aktualnych musi być zgodna co do liczby argumentów i ich typów z odpowiednią listą parametrów formalnych
— moduły obsługi przerwań <idh>, występujące w instrukcji **mask**, muszą być zadeklarowane w procesie wykonującym tę instrukcję
— w instrukcji **interrupt** <idvp> . <idh> ...;, <idh> musi być identyfikatorem modułu obsługi przerwania zadeklarowanego w procesie wskazywanym przez <idvp>.

W każdym z procesów można odczytywać wartości standardowej zmiennej **father**, wskazującej na ojca dynamicznego danego procesu oraz zmiennej **this** wskazującej na proces, w którym ona występuje (tekstowo).

W każdym zaś module obsługi przerwania odczytać można wartość zmiennej **sender** wskazującej na proces, który spowodował obsługiwane przerwanie. Ponadto przyjmujemy, że nazwy zmiennych i przerwań mogą być identyczne. Właściwość ta nie prowadzi do nieporozumień, a zwiększa czytelność programów zapisanych w PCI.

Poniższe reguły widoczności identyfikatorów precyzują sens używanego przez nas sformułowania p r o c e s y r o z p r o s z o n e:

- identyfikatory modułów obsługi przerwań są widoczne dzięki dostępowi przez kropkę
- identyfikatory typów procesów są widoczne zgodnie ze zwykłymi regułami dla bloków, przy czym rolę bloków odgrywają tu procesy
- zmienne widoczne są jedynie w tym procesie (module obsługi przerwania), w którym są zadeklarowane.

SEMANTYKA

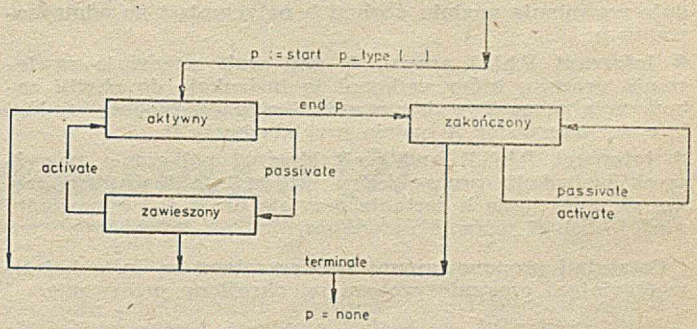
Opiszemy teraz krótko semantykę prezentowanego systestemu procesów. W tym celu wprowadźmy dwa pojęcia — stanu procesu i stanu przerwania w procesie:

— każdy proces różny od none (none oznacza proces nieistniejący) może być w jednym z trzech stanów: aktywny (gdy wykonuje swoje obliczenia), zawieszony (gdy jego obliczenia są wstrzymane) lub zakończony (gdy zakończył wykonywanie swoich instrukcji)

— każde przerwanie w danym procesie może być albo zamaskowane (i wówczas jest przez proces ignorowane), albo nie zamaskowane (wtedy nadesłane przerwanie powoduje natychmiastowe wywołanie odpowiedniego modułu obsługi).

Nie zamaskowane przerwania są obsługiwane zarówno przez procesy aktywne, zawieszone, jak i zakończone.

Początkowo system składa się z jednego procesu (o najbardziej zewnętrznej specyfikacji). Zmiany stanu systemu mogą powodować następujące instrukcje (por. rys.):



- p : = **start** p_type (parametry aktualne) — która powoduje wyliczenie wartości parametrów aktualnych i przekazanie ich do nowego obiektu procesu typu p_type wskazywanego od tego momentu przez zmienną p. Wszystkie przerwania w p są początkowo zamaskowane. Zmienna **father** procesu p wskazuje na proces, który wykonał p : = **start** p_type(...)
- **passivate** — która powoduje zawieszenie wykonującego ją procesu (wykonana w procesie zakończonym jest instrukcją pustą)
- **activate** — która powoduje aktywację wykonującego ją procesu (wykonana w procesie zakończonym jest instrukcją pustą)
- **terminate** — która powoduje zakończenie wykonującego ją procesu i usunięcie go z systemu
- **end** — która powoduje zakończenie działania procesu.

We wszystkich opisywanych instrukcjach występują jedynie parametry wejściowe (ang. input parameters).

Stan przerwania w danym procesie P może się zmienić po wykonaniu instrukcji powrotu z modułu obsługi przerwania (**return** lub **end**) albo instrukcji **mask**. W pierwszym przypadku stan przerwań w procesie P powraca do stanu, jaki był w momencie wywołania kończonego modułu obsługi. Instrukcja **mask** i1, ..., in powoduje, że przerwania i1,..., in są zamaskowane, natomiast przerwania różne od i1, ..., in — nie są. **Mask all** powoduje zamaskowanie wszystkich przerwań.

Najważniejsza w naszej prezentacji jest instrukcja przerwania procesu: **interrupt** P.h (parametry aktualne) **answer** b. Na potrzeby artykułu przyjęto, że b jest zmienną logiczną, umożliwiającą stwierdzenie, czy przerwanie h zostało przyjęte przez P czy też nie (wygodne jest oczywiście rozszerzenie liczby różnych odpowiedzi w ten sposób, aby określały one przyczynę zignorowania przerwania, np. „przerwanie zamaskowane", „awaria" czy „brak odpowiedzi"). Wykonanie instrukcji **interrupt** rozpoczyna się od

wyliczenia wartości parametrów aktualnych, a następnie:
— jeśli h jest zamaskowane w P, to jedynym wynikiem działania **interrupt** jest podstawienie na zmienną b wartości **false**
— jeśli h nie jest zamaskowane w P, to P zostaje przerwany, parametry aktualne **interrupt** oraz sterowanie P przekazywane są do modułu obsługi h; na zmienną b podstawiana jest wartość **true**, po czym oba procesy kontynuują obliczenia. Zakończenie działania modułu obsługi przerwania może nastąpić albo po wykonaniu instrukcji **return** lub **end** (wówczas sterowanie powraca do miejsca, w którym proces został przerwany), albo po wykonaniu **terminate** (co, jak zaznaczono wcześniej, powoduje usunięcie procesu z systemu).

Zakładamy, że proces nie może zostać przerwany podczas wykonywania instrukcji prostych (**activate, return, mask** itd.) oraz pomiędzy nimi, jeśli separatorem jest &.

ROZSZERZENIA MECHANIZMU KOMUNIKACJI

Zaprezentowane w tej części artykułu rozszerzenia mechanizmu przerwań można zaprogramować w „czystym" języku PCI. Występują one jednak na tyle często w programach, że włączenie ich do języka uważamy za uzasadnione.

Trzy pierwsze z proponowanych instrukcji stanowią modyfikację **interrupt**. I tak:

- **interrupt** P.h(...)! **answer** b różni się od **interrupt** P.h (...) **answer** b tylko wtedy, gdy przerwanie h jest zamaskowane w P; wówczas nie jest ono ignorowane, lecz powoduje wywołanie modułu obsługi h natychmiast po odmaskowaniu h
- **interrupt** P.h(...) **answer** b **and wait** powoduje zawieszenie procesu, który wykonał tę instrukcję do chwili zakończenia przez P wykonywania instrukcji modułu obsługi h, o ilę przerwanie h zostało przyjęte przez P
- **interrupt** P.h(...)! **and wait** stanowi połączenie powyższych instrukcji: proces, który wykonał tę instrukcję czeka, aż przerwanie h zostanie przyjęte przez P i odpowiedni moduł obsługi zostanie zakończony.

Ostatnimi proponowanymi rozszerzeniami są instrukcje umożliwiające czekanie procesu na określone przerwanie:

- **wait for** h — powoduje oczekiwanie procesu do chwili nadejścia przerwania h. Podczas oczekiwania wszystkie przerwania różne od h są odrzucane
- **wait for** h **from** P — jest modyfikacją **wait for** h i powoduje oczekiwanie procesu na przerwanie h, którego nadawcą jest proces P.

W obu powyższych instrukcjach moduł h powinien być zadeklarowany w procesie, który wykonuje te instrukcje.

PRZYKŁADY

W celu przybliżenia czytelnikom języka PCI, przytoczymy na zakończenie artykułu dwa przykłady. Pierwszy z nich opisuje schemat prostego systemu, mogącego znaleźć zastosowanie w sterowaniu produkcją, eksperymentami naukowymi itp. W skład systemu wchodzą przyrządy pomiarowe i główny komputer. W każdy z przyrządów pomiarowych wbudowany jest prosty mikroprocesor sprawdzający, czy wartości pomiarów nie wykraczają poza dopuszczalny zakres. Jeśli wykraczają, to zgłaszane jest do głównego komputera przerwanie „alarm":

Drugi przykład pokazuje, w jaki sposób można programować w języku PCI nowe struktury danych. Proces queue opisuje kolejkę dowolnych procesów z określonymi na niej zwykłymi operacjami: wstawiania procesu na koniec ko-

Przykład 1

```
process system;
var M : main,
    D1,D2,D3,D4 : device;
 process main;
 (*proces opisujący zachowanie głównego komputera*)
  .....
  handler alarm (value : real, number : integer) ;
  (* reaguje na sytuację alarmową, wypisując odpowiedni
    komunikat i podejmując właściwe akcje*)
  begin mask all;
   write(" alarm w urządzeniu o numerze", number,
         " spowodowany przez wartość", value);
   .....(*inne akcje, np. wyłączenie urządzenia*)
  end alarm;
begin
 mask ; (*odmaskowanie wszystkich przerwań*)
 .....(*dowolne obliczenia głównego komputera*)
end main;
process device (number : integer, M : main) ;
(*number - numer logiczny urządzenia*)
var value : real; (*wartość pomiaru*)
begin
 do value:= wartość kolejnego pomiaru ;
    if value wykracza poza dopuszczalny zakres
    then interrupt M.alarm(value, number)! fi
 od (*do ... od oznacza while true do ... od *)
end device;
begin (*instrukcje procesu system*)
 M := start main;
 D1 := start device(1,M);
 D2 := start device(2,M);
 D3 := start device(3,M);
 D4 := start device(4,M);
end system;
```

Przykład 2

```
process queue;
var front, tail, aux : qelem,
    result : process;
 process qelem (p : process) ;
 (*opisuje element kolejki*)
 var next : qelem;
  handler trm; begin terminate end trm;
  handler value_of_next;
 (*przekazuje nadawcy przerwania wartość zmiennej next*)
  begin interrupt sender.value_of_next (next)
  end value_of_next;
  handler modify(x : qelem ) :
  begin next:= x end modify;
  handler result;
  begin interrupt sender.result(p) end result;
begin mask ; end qelem;
handler into(x: process) ; (*wstawia x do kolejki*)
begin mask all;
  aux:= start qelem(x) ;
  if front = none then front, tail := aux
  else interrupt tail.modify(aux) and wait;
       tail:= aux
  fi
end into;
handler out; (*usuwa pierwszy element z kolejki*)
begin mask all;
  if front =/= none then
   interrupt front.value_of_next & wait for value_of_next;
   (*teraz aux = front.next*)
   interrupt front.trm and wait; (*usunięcie procesu front*)
   front:= aux
  fi
end out;
handler first;
begin mask all;
  if front =/= none then
   interrupt front.result & wait for result;
  else result:= none fi;
       interrupt sender.first (result)
       end first;
       handler empty;
       begin interrupt sender.empty(front = none)
       end empty;
       handler result (x : process) ,
       begin result := x end result;
       handler value_of_next (x : qelem)
       begin aux:= x end value_of_next;
     begin mask ; end queue;
```

lejki (into), usuwania pierwszego procesu z kolejki (out), pobierania pierwszego elementu z kolejki bez usuwania go (first) oraz sprawdzania czy kolejka jest pusta (empty):

A oto przykład wykorzystania tak zaprogramowanej kolejki:

```
process user;
var empty : boolean,
    first : process,
    q : queue;
handler first(x: process);
begin first:= x end first;
handler empty(x : boolean)
begin empty:= x end empty;
begin mask ;
  q:= start queue; .....
  interrupt q.into(this) !;
  .....
  interrupt q.empty! & wait for empty;
  if empty then ..... else ..... fi;
  .....
end user;
```

ROMAN ŻELĄZNY
Warszawa

LITERATURA

[1] Brinch Hansen P.: Distributed Processes: A Concurrent Programming Concept. CACM, vol. 21, No. 11, 934—941, 1978
[2] Brinch Hansen P.: Podstawy systemów operacyjnych, WNT, Warszawa, 1979
[3] Dannenberg R. B.: AMPL: Design, Implementation and Evaluation of a Multiprocessing Language, Dept. of Comp. Sci., Carnegie Mellon University, March 1981
[4] Hoare C.A.R.: Communicating Sequential Processes. CACM, vol. 21, No. 8, 666—677, 1978
[5] Hoare C.A.R.: Monitors: an Operating System Structuring Concept. CACM, vol. 17, No. 10, 549—557, 1974
[6] Introduction to CHILL, the CCITT High Level Programming Language. CCITT, May 1980
[7] Kreczmar A. (red.): Report on the LOGLAN 84 Programming Language, Raport IIUW, Wraszawa, 1984
[8] Petermann U.: System plików — aksjomatyczna specyfikacja i analiza możliwości zrealizowania systemu w języku LOGLAN. Praca doktorska złożona w IIUW, 1984
[9] Petermann U., Szałas A.: On Distributed Processes Communicating by Interrupting Signals, Raport IIUW, 137, Warszawa, 1984
[10] Reference Manual for the ADA Programming Language. Proposed Standard Document, US Dept. of Def., July 1980
[11] Szałas A.: Prace eksperymentalne nad językiem programowania właściwym do tworzenia systemów operacyjnych. Praca doktorska, IIUM, Warszawa, 1984
[12] Szałas A., Szczepańska-Wasersztrum D.: Exception Handling in Parallel Computations, Raport IIUW, 139, Warszawa, 1984.

Narzędzia inżynierii oprogramowania (2)

Poprzednią część tego przeglądu zakończyliśmy opisem systemu PSL/PSA. Ma on wielu kontynuatorów i następców. Jednym z nich jest system DAS, który został przekształcony w system AIDES, używany przez firmę HUGHES AIRCRAFT COMPANY [28]. Różni się od niego system RDL, zbudowany i wykorzystywany przez SPERRY UNIVAC [18]. Został on wygenerowany za pomocą systemu META projektu ISDOS. Szczególną właściwością języka RDL jest istnienie obiektów explicite zdefiniowanych dla pozostałych faz cyklu produkcyjnego oprogramowania. Ta właściwość pozwala bardziej sytematycznie śledzić i analizować cele, koncepcje, implementacje oraz testy dla celów nadzoru i weryfikacji.

Bardziej znanym i lepiej opisanym w literaturze jest tzw. program inżynierii wymagań użytkowych (Software Requirements Engineering Program, SREP) lub metodologia (SREM) [1, 2, 3, 8] stworzone przez Ośrodek Zaawansowanej Technologii Systemów Obrony Przeciwrakietowej Armii Stanów Zjendoczonych (Ballistic Missile Defense Advanced Technology Center). Zawdzięczają one wiele koncepcjom projektu ISDOS, a pewne ich fragmenty są wręcz pochodnymi PSL/PSA. Ich język i analizator to RSL oraz REVS, zaś baza danych — ASSM.

Główne nowe aspekty systemu SREM można scharakteryzować następująco. RSL jest językiem rozwojowym w tym sensie, że pewne podstawowe rodzaje typów zostały wbudowane w system. Można ich użyć do definiowania dodatkowych, potrzebnych pojęć językowych. Tymi bazowymi typami są: **elementy, atrybuty i relacje.** Te trzy podstawowe pojęcia odpowiadają pojęciom PSL. Główna różnica polega na tym, że w RSL można definiować nowe typy pojęć. Stałymi pojęciami RSL są struktury, reprezentacje dwuwymiarowych grafów przepływów, tzw. sieci wymagań użytkowych (w skrócie R-net — sieci-w). Składają się one z węzłów, które określają operacje przetwarzania, oraz linii, które je łączą. Podstawowymi węzłami są tzw. węzły ALFA, które specyfikują funkcjonalne etapy przetwarzania, oraz podsieci, które z kolei specyfikują przepływy przetwarzania na niższym poziomie hierarchii. Można powiedzieć, że podsieć jest to węzeł ALFA, z wewnętrzną strukturą przetwarzania. Bardziej złoożne sytuacje przepływu przetwarzania można rozwinąć w RSL przy użyciu węzłów złożonych, które zbierają lub rozprowadzają różne ścieżki przetwarzania. Złożonymi węzłami są „i", „lub" i „dla każdego". Składnia struktur sieci-w jest podobna do składni wielu strukturalnych języków programowania i wymusza na użytkowniku dyscyplinę przez używanie ustalonego zestawu elementarnych operacji przepływowych. Takie podejście pozwala na wyrażenie wymagań użytkowych nie tylko pod względem funkcjonalnym, lecz i wykonawczym, dzięki nakładanym więzom na ścieżki (przepływy) przetwarzania.

Stwierdzenia (asercje) w języku RSL są tłumaczone przez translator RSL i wprowadzane do bazy ASSM. Struktura ASSM jest różna od struktury bazy danych w PSL/PSA. Umożliwia ona uzyskanie niezależności między językiem wejściowym RSL a narzędziami analizy — REVS. Dzięki temu możliwy jest rozwój języka RSL oraz duża swoboda w budowie systemu REVS. Narzędzia mają dostęp do ASSM i w żaden sposób nie są zależne od składników RSL, które można modyfikować i rozbudowywać wraz z ewolucją SREM.

Wśród różnorodnych narzędzi REVS należy wymienić narzędzie interakcyjnego generowania sieci-w, które pozwala użytkownikowi na ich wprowadzenie, modyfikowanie i przedstawienie w sposób graficzny. Graficzna i językowa reprezentacja sieci-w są całkowicie równoważne i wymienne między sobą. Ważnym narzędziem REVS jest także automatyczny generator symulacyjny, który na podstawie sformułowań wymaga użytkowych systemów zapisanych w ASSM generuje symulatory systemu. Są one oparte na zdarzeniach i uruchamiane przez zewnętrzne sygnały.

Istnieja dwa rodzaje symulatorów. Jeden z nich wykorzystuje funkcjonalne modele procesów przetwarzania, które służą do sprawdzania zgodności ogólnego przepływu przetwarzania z wyższymi poziomami wymagań systemowych. Drugi rodzaj symulatorów wykorzystywany jest do określania zestawu algorytmów systemowych, które charakteryzują się żądaną dokładnością i stabilnością. Obu rodzajów

symulatorów używa się do badania oddziaływań dynamicznych wewnątrz systemu oraz weryfikacji kryteriów. Symulatory napisane są w języku PASCAL. Niezależnie, istnieje grupa narzędzi, które sprawdzają statycznie (bez symulacji) zgodność i kompletność specyfikacji wymagań użytkowych. Wykrywają one luki w sformułowanych wymaganiach użytkowych, w przepływach przetwarzania i manipulacjach danymi.

Wydaje się, że SREM spełnia większość celów związanych z produkcją ścisłych, sprawdzalnych wymagań użytkowych, formułujących funkcje przetwarzania danych oraz wymagań wykonawczych. W ramach tych prac zbadano różne alternatywy kompozycyjne projektowania: metodę grafów weryfikacyjnych, metodę sieci Petriego oraz podejście maszyn skończonych. Ostatnie podejście wykorzystuje model grafów jako relację między funkcją a jej podfunkcjami składowymi. Zautomatyzowane narzędzia (REVS) przeniesiono na komputer CDC i czas ich pracy został zredukowany około stukrotnie, dzięki czemu uzyskano wydajne narzędzie inżynierii oprogramowania. Są one obecnie sprawdzonymi, pracującymi samodzielnie narzędziami do definiowania i sprawdzania wymagań użytkowych. Gładkie przejście do wymagań użytkowych do projektowania jest jednak wciąż aktualnym przedmiotem badań. Dodając fazę projektowania do SREM uzyskuje się System Budowy Systemów (SDS). Faza projektowania SDS odwzoruje procesy przetwarzania na architekturę oprogramowania opisaną w języku projektowania procesów, zapewnia narzędzia weryfikacji i sprawdzania projektu, ewolucji projektu w kompletny program (kod) oraz umożliwia unikanie lub wczesne wykrywanie błędów.

Tak powstała metodologia projektowania procesów (PDM) [11] oraz język projektowania procesów PDL2 [19] — zapewniają zintegrowane podejście do zarysowanych problemów. Metodologia dostarcza procedur podziału wymagań użytkowych na zadania oraz definiuje techniki koordynacji wymagań niezbędnych dla uzyskania pożądanych odpowiedzi systemu. Przy jej użyciu w systemie projektowania zastępującego, specyfikuje się wyznaczanie zadań procesorowych, sprzężenia zadaniowe oraz charakterystykę kontroli zadań. Na narzędzia projektowania procesów, niezbędne dla realizacji omawianej metodologii, składają się narzędzia do zarządzania biblioteką i kontroli konfiguracji oprogramowania, narzędzia do konstrukcji procesów, do kompozycji modeli funkcjonalnych oraz symulacji na poziomie algorytmów tej biblioteki, do automatycznej rekompilacji zmodyfikowanych kodów oraz do specyfikacji technik gromadzenia danych. Język PDL2 oparto na PASCALU, a w uzupełnieniu takich cech PASCALA, jak ścisła kontrola typów danych, struktura blokowa oraz kontrola zakresu zmiennych, zawiera on podstawowe pojęcia synchronizacji zadań, założeń logicznych, zmiennej wymiarowości tablic oraz sytuacji wyjątkowych. Doświadczenia uzyskane w użytkowaniu PDL2 były wykorzystane przy sformułowaniu wymagań w odniesieniu do ADY.

Rozwój SDS umożliwił wielorakie postępy w inżynierii oprogramowania [?]:
— zostały opracowane języki, narzędzia i metodologie dla różnych faz procesu produkcyjnego oprogramowania
— udowodniono, że można wykrywać i likwidować błędy we wczesnych fazach cyklu produkcyjnego oprogramowania, przyczyniając się w ten sposób do zwiększenia jego niezawodności, zmniejszenia kosztów produkcji oraz ryzyka złego wykonawstwa
— zarówno doświadczenia systemu SREM, jak i PDM wykazały, że symulacje można przeprowadzać za pomocą zautomatyzowanych narzędzi wprost ze specyfikacji wymagań użytkowych oraz specyfikacji projektowych, zmniejszając w ten sposób ryzyko błędnej interpretacji tych specyfikacji przez programistę
— użycie obu metod wykazało, że metodologię można zdefiniować za pomocą zobiektywizowanych celów częściowych; zautomatyzowane narzędzia mogą być użyte do weryfikacji, czy cele te zostały spełnione, to zaś pozwala na realistyczne śledzenie postępu prac.

Doświadczenia zdobyte przy realizacji SDS doprowadziły do zidentyfikowania pewnej liczby zagadnień, które są ważne dla przyszłych środowisk inżynierii oprogramowania:

• Środowiska inżynierii oprogramowania winny być zorganizowane na najwyższym poziomie systemowym i uwzględniać problem dekompozycji oraz podziału funkcjonalnych i wykonawczych wymagań w stosunku do zagadnień przetwarzania. Bez tych zasad powstaje ryzyko „rozwiązywania niewłaściwego problemu".

• Użyteczność metolodogii znacznie wzrasta, gdy istnieją gładkie przejścia między fazami wymagań systemowych (użytkowych), projektowania i implementacji. Zunifikowany model tych faz jest konieczny do uzyskania ich pełnej integracji (dopóki problem ten nie jest rozwiązany, mogą zachodzić poważne niewydolności związane z przejściami między fazami).

BARDZIEJ WSPÓŁCZESNE PODEJŚCIA

Prace badawcze dotyczące realizowalności środowisk produkcji oprogramowania były prowadzone przez wiele różnych grup i instytucji. Omówimy teraz doświadczenia z prac opartych na bardziej nowoczesnych podejściach. Szczególnie ważnym, nowym pojęciem ostatniej dekady jest pojęcie abstrakcji. Zostało no wprowadzone przez pionierskie prace Parnasa na temat modularyzacji [23] oraz technik specyfikacji [22], jak również wynikło z pojęcia klasy w języku SIMULA [7]. Zostało później rozwinięte przez Guttaga, Liskow, Zilles i innych [13, 14, 20, 21, 30] i jest podstawą współczesnych prac badawczych i rozwojowych w inżynierii oprogramowania.

Dobrym przykładem takich prac rozwojowych jest hierarchiczna metoda tworzenia, w skrócie HDM (Hierarchical Development Methodology), opracowana w Stanfordzkim Instytucie Badawczym (SRI) [10, 12, 19, 26]. HDM zakłada, że system ma strukturę pionową i poziomą. Dzieli się on przede wszystkim na oddzielne poziomy albo abstrakcyjne maszyny (podział pionowy). Każdy poziom dostarcza zestaw „urządzeń" (procesorów) następnemu, wyższemu poziomowi w hierarchii. „Urządzenia" (procesory) udostępniane przez najwyższy poziom są w bezpośredniej dyspozycji użytkownika systemu. Najniższy poziom nazywa się maszyną bazową — spoczywa na niej cała hierarchia. Maszyna bazowa dostarcza wszystkie te urządzenia czy procesory, które projektant uważa za dane. Może to być podstawowy sprzęt, maszyna abstrakcyjna reprezentowana przez system operacyjny lub też maszyna hipotetyczna w postaci procesora języka wysokiego poziomu (np. „maszyna" języka PASCAL).

Maszyna abstrakcyjna HDM składa się ze zbioru struktur danych wewnętrznych, które definiują jej stan, oraz zbioru operacji, które mogą działać na te dane i modyfikować je (modyfikując jej stan). Uruchomienie operacji wywołuje transformację stanu, a uruchomienie programu wywołuje sekwencję transformacji stanu. Poza podziałem pionowym na poziomy akstrakcyjne maszyn, każdy system dowolnej wielkości musi być podzielony wewnątrz każdego poziomu na oddzielne jednostki lub moduły. Jeden ze sposobów modularyzacji maszyny abstrakcyjnej polega na podziale według „urządzeń" (procesorów). Każde „urządzenie" (procesor) zamknięte jest w osobnym module. Wtedy efekty ewentualnych przyszłych zmian mogą być zminimalizowane. Jest obecnie powszechnie przyjętą regułą, że zachowanie zewnętrzne każdego modułu musi być odseparowane od jego wewnętrznych szczegółów. Zatem specyfikacja modułu definiuje zewnętrzne zachowanie się modułu bez ujawniania jego implementacji. Niezależna specyfikacja umożliwia niezależną implementację oraz sprawdzanie. W ten sposób struktura systemu może być zmodularyzowana i proces jego budowy może być realizowany krok po kroku.

Sercem HDM jest jego język specyfiki — SPECIAL (język specyfikacji i stwierdzień, ang. specification and assertion language). Znaczna część języka SPECIAL została sformalizowana w ramach teorii Bovera-Moore'a [4, 5] i jest bazą systemów weryfikacyjnych SRI. Moduł HDM składa się ze struktur danych oraz z określonych na nich operacji. Struktury danych definiujące stan scharakteryzowane są przez ich funkcjonalne zachowanie (mówi się więc o funkcjach stanu). Specyfikacja funkcji stanu polega na podaniu jej sygnatury oraz ograniczeń wartości początkowych. Zmianę stanu opisuje się przez powiązanie wartości funkcji stanu po i przed wykonaniem operacji. Wynikowa wartość zdefiniowana jest przez warunki (ograniczenia), które musi spełnić. W języku SPECIAL można definiować sytuacje wyjątkowe za pomocą tzw. warunków wyjątkowych. Sytuacja wyjątkowa wyklucza zmianę stanu.

Ze względów historycznych — konstrukcje języka SPECIAL opisują nie operacje i funkcje stanu, lecz raczej tzw. V-funkcje, O-funkcje oraz OV-funkcje wywodzące się z terminologii Parnasa [22]. Operacjami zmieniającymi stan są O- i OV-funkcje. OV-funkcja może zmieniać stan i udostępniać wartość, natomiast O-funkcja może tylko zmieniać stan. V-funkcja nie może zmieniać stanu, a jedynie dostarcza informacji o stanie modułu.

Poza językiem specyfikacji modułów SPECIAL, którego używa się również do określania funkcji odwzorowania, HDM używa innych języków. Języka HSL (język specyfikacji hierarchii) używa się do opisu konstrukcji maszyn abstrakcyjnych i całych systemów. ILPL (pośredni język programowania) jest językiem programowania maszyn abstrakcyjnych. Używa się go do zapisywania decyzji związanych z implementacją modułów. Używa się języków pozwalających na wykonanie programów (np. PASCAL czy MODULA).

Silną stroną metody HDM jest jej zdolność weryfikacji, którą zawdzięcza swej sformalizowanej strukturze. Istnieją dwie możliwe formy weryfikacji: projektowa oraz implementacyjna. Weryfikacje projektowe mają na celu wykazanie, że specyfikacja systemu posiada określone właściwości funkcjonalne. Weryfikacja implementacyjna ma na celu dowiedzenie, że implementacja programistyczna systemu spełnia wymagania specyfikacji. Analizatory specyfikacji składają się ze zwykłych analizatorów składni (ang. parser), weryfikatorów typów oraz specjalnie oprogramowanych drukarek dla specyfikacji formułowanych w językach SPECIAL, HSL i ILPL. Weryfikują one spełnienie różnych kryteriów zgodności wewnętrznej, wynikających z warunków, które muszą być spełnione przez indywidualne specyfikacje cząstkowe — aby były spójne ze specyfikacją całego systemu.

Weryfikator międzypoziomowego systemu zabezpieczeń użytkowany jest do sprawdzenia, czy projekt przedstawiony przez zestaw specyfikacji modułów w języku SPECIAL spełnia szczegółowe wymagania systemu zabezpieczeń. Wykorzystuje on rozbudowaną wersję automatycznego dowodzenia twierdzeń opracowaną przez Boyera i Moore'a. Podejście HDM do weryfikacji implementacji opiera się na metodzie Floyda-Hoare'a, w której sprawdza się czy program spełnia stwierdzenia (asercje) wejściowe i wyjściowe. Pełny opis metody dowodów hierarchicznych zawarty jest w pracy [24]. Są również zrealizowane weryfikatory języków MODULA i PASCAL. Na podstawie udowodnionej w praktyce stosowalności metodyki HDM do projektowania modułów, specyfikowania i weryfikacji, przygotowuje się „drugą generację" metodologii HDM. SPECIAL ma być rozbudowany, aby nie pozostać w tyle za współczesnymi osiągnięciami w dziedzinie języków i ma uwzględniać jednakowo abstrakcyjne podejście i do danych, i do procedur. Są plany zintegrowania języka ADA z HDM. Narzędzia ukierunkowane na język ADA pozwolą na mechaniczne sprawdzanie właściwości modułów HDM implementowanych w tym języku.

Specyfikacje w tej metodzie mają być ukierunkowane nie tylko na cele projektowania, lecz w dużo większym stopniu na przedstawienie (formułowanie) wymagań użytkowych. Istnieje tendencja do wykorzystania osiągnięć metody SREM, jak np. wyrażenia sieci-w w języku SPECIAL, a także innych podejść tej metodologii w postaci zaawansowanych koncepcji bazy danych dla zapisu decyzji projektowych, pielęgnacji wersji i konfiguracji oraz skoordynowania różnych narzędzi projektowania i analizy w jedno środowisko. Planuje się zbadanie przydatności koncepcji biblioteki modułów dla ułatwienia użytkownikowi lepszego skoordynowania wymagań użytkowych, specyfikacji, implementacji i dokumentacji (baza danych specyfikacji modułów). Bardzo interesujące będzie śledzenie dalszego rozwoju metody HDM, w kierunku jednorodnego środowiska budowy oprogramowania, wykorzystującego szeroko rozbudowany język, zawierający konstrukcje niezbędne dla opisu wymagań użytkowych, oraz posiadającego potężne, skomputeryzowane możliwości weryfikacyjne.

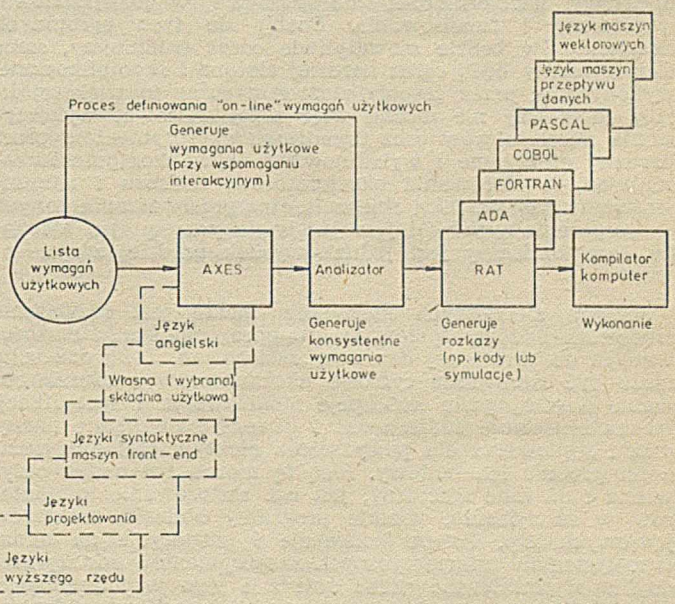
Ciekawe jest również podejście zaproponowane przez firmę HIGHER ORDER SOFTWARE, znane pod nazwą HOS [15, 16, 17]. Teoria HOS opiera się na zbiorze sześciu aksjomatów, opisujących właściwości hierarchicznych struktur systemowych, zwanych kontrolerami lub modułami, które realizują funkcje. Moduły ulokowane są w węzłach położonych bezpośrednio wyżej w drzewie logicznym w stosunku do funkcji, które kontrolują. Moduł odpowiada za realizację funkcji, które kontroluje.

Każda funkcja otrzymuje bezpośrednio lub pośrednio od kontrolera informację wejściową i dostarcza mu wyjściową. Systemy skomponowane są z map kontrolnych HOS. Dokonuje się tego przy użyciu języka specyfikacji HOS — AXES. Jest to dobrze zdefiniowany, kompletny język, który może być analizowany przez komputer. Zawiera on mechanizm definiowania typów danych (dla identyfikacji obiektów), funkcji (dla wiązania ze sobą obiektów różnych typów) i struktur (dla wiązania ze sobą różnych funkcji).

Z aksjomatów HOS wyprowadza się zbiór trzech prymitywnych struktur kontrolnych. Te struktury identyfikują schematy kontrolne na zbiorach obiektów. W języku AXES istnieje mechanizm do zdefiniowania algebry dla każdego wyróżnionego zbioru obiektów. W celu stworzenia systemu, definiuje się nowe struktury kontrolne przy użyciu struktur prymitywnych lub innych struktur nieprymitywnych. Mając bibliotekę struktur kontrolnych, typów danych, operacji oraz złożonych operacji, można tworzyć nowe definicje w języku AXES. Definicje te można wprowadzać do komputera w reżimie interakcyjnym — graficznie lub tekstowo.

Postać komputerowa tych rozwiązań pozwala na to, by analizator mógł automatycznie sprawdzić ich kompletność logiczną, wykryć błędy w sprzężeniach, konflikty czasowe lub konflikty typów, błędy w definiowaniu danych oraz struktur kontrolnych. Wykryte błędy przedstawione są analitykowi systemu, który może je korygować wykorzystując edytor graficzny.

Specyfikacje w języku AXES, sprawdzone pod względem logicznym przez analizator, przekazywane są automatycznie do narzędzia podziału zasobów (RAT, ang. Resource Allocation Tod). RAT odwzorowuje sprawdzone specyfikacje funkcjonalne na środowisko docelowe, generując kod dla maszyny docelowej. Jest to automatyczny programista przekształcający specyfikacje otrzymane od analizatora bezpośrednio na program źródłowy lub docelowy dla wybranego komputera. Programy te są ściśle zgodne ze sprawdzoną specyfikacją. System HOS.USE.IT dostarcza środków do wykonywania programów na dowolnym etapie ich budowy. Nie w pełni skompletowane programy mogą być wykonane przez symulację brakujących jego części. System wzywa analityka do wprowadzenia danych dotyczących niewykończonych części programu. Te możliwości symulacyjne dają analitykowi oraz użytkownikowi szansę modyfikacji i usprawnienia systemu pod względem wymagań funkcjonalnych i wykonawczych (rys.).



W ten sposób metodologia HOS zezwala na całkowicie automatyczną implementację dowolnego, definiowalnego systemu ze specyfikacji, które można wprowadzić do komputera i sprawdzić automatycznie pod względem logicznym. Specyfikacje mogą wykorzystywać dowolną składnię, którą można transformować na język specyfikacyjny AXES za pomocą graficznego, interakcyjnego edytora. Nie potrzeba żadnej ludzkiej interwencji, by przekształcać specyfikację w kod oraz uzyskać niezbędną dlań dokumentację. HOS ma cechy, które w sposób komplementarny uzupełniają istniejące metodologie oraz dowodzi, że automatyzacja implementacji jest realizowalna w praktyce.

Przegląd ten wskazuje, że idea środowisk budowy oprogramowania nie jest zbyt daleka od praktycznej realizacji. Wiele koncepcji oraz czynności, narzędzi do definiowania

wymagań użytkowych, specyfikacji, różnych faz projektowania, implementacji, weryfikacji i pielęgnacji zostało opracowanych i sprawdzonych w realizacjach eksperymentalnych, dowodząc ich praktycznych możliwości. Nie jesteśmy zbyt daleko od realizacji takich systemów, jak np. COMPASS, obejmujących prawie wszystkie koncepcje współczesnej inżynierii oprogramowania [25].

Czy istnieje możliwość jeszcze innego podejścia? Interesujące stanowiska, poglądy i propozycje pojawiają się w literaturze w sposób ciągły. Warto zapoznać się np. z artykułem Winograda [29] na temat systemów programistycznych wysokiego poziomu oraz z artykułem Wassermana i Gutza [27] na temat przyszłości programowania. Wiele interesujących rzeczy dzieje się w dziedzinie specyfikacji algebraicznych. Szczególnie wart uwagi jest projekt pod nazwą ACT na temat specyfikacji algebraicznych, realizowany w Berlinie [9], lecz jest to już materiał na inny artykuł.

WNIOSKI

Systemy budowy oprogramowania konstruuje się i stosuje z powodzeniem w wielu dziedzinach przemysłu i badań naukowych, polepszając produkty oprogramowania pod względem kosztów, niezawodności, problemów pielęgnacji oraz efektywności.

Powstaje więc pytanie, które ze współczesnych i przyszłych narzędzi inżynierii oprogramowania oraz systemów mogłyby być zaadaptowane i zastosowane na terenie przykładowej dziedziny jaką jest fizyka, a szczególnie — w eksperymentach fizycznych na wiązce cząstek z akceleratorów (zarówno wysokiej, jak i pośredniej energii). Ta klasa eksperymentów wydaje się być w chwili obecnej dobrze zdefiniowana, z tendencją do stabilności w najbliższej przyszłości, a ich przeprowadzanie jest oparte w sposób istotny na intensywnym wykorzystywaniu komputerów. Dokonuje się tego jednak w sposób mniej lub bardziej spontaniczny, nieskorelowany, nieskoordynowany. Z drugiej strony — eksperymenty na wiązkach cząstek akceleratorowych są wykonywane przez coraz większe zespoły, złożone z pracowników wielu laboratoriów i taka tendencja będzie się utrzymywać w sposób zbieżny z tendencją do budowania coraz większych akceleratorów, na zasadzie międzynarodowej współpracy i finansowania. Dostęp do tych ogromnych akceleratorów będzie niewątpliwie coraz trudniejszy, same eksperymenty będą coraz bardziej złożone i skomplikowane, wykonywane przez zespoły złożone z coraz to większej liczby osób. Czy nie można więc zoptymalizować wykorzystania akceleratorów oraz organizacji eksperymentów przez bardziej efektywne i wyrafinowane użycie komputerów — zarówno do sterowania akceleratorów, nadzoru i administrowania wiązek, jak i zbierania oraz przetwarzania danych eksperymentalnych? Podjęto już wiele prac w tym kierunku, lecz konieczny jest jednak wysiłek bardziej zorganizowany.

Jednym z istotnych kierunków badań jest rozważenie środowisk budowy systemów lub oprogramowania dostosowanych do określonej dziedziny zastosowań. Taką dziedziną może być sterowanie cyfrowe akceleratorów i administrowanie wiązką, gdzie koncepcje modelowania i specyfikacji można sensownie zdefiniować i utworzyć niezbędne narzędzia i środowiska dla powyższenia efektywności i użyteczności procesu ich budowy. Ideę tę można rozszerzyć oczywiście i na inne dziedziny, jak np. zastosowanie komputerów w zarządzaniu, wielkie programy obliczeń reaktorów jądrowych, itp. Pewne koncepcje i propozycje na temat tzw. systemów informatycznych eksperymentów zostały niedawno sformułowane przez autora i jego współpracowników [6]. Można je podsumować następująco: środowiska budowy oprogramowania pomyślane dla tych eksperymentów mogłyby być nie tylko podstawą do budowy niezbędnego oprogramowania, lecz przez rozproszone urządzenia komputerowe mogą stwarzać możliwości bardziej skoordynowanego wykorzystania danych eksperymentalnych, programów i metod opracowania danych, eksperymentów logicznych, ponowej oceny publikowanych wyników, itp. Takie prace badawcze w fizyce, być może zorganizowane w ramach projektu LEP [1]), są warte przemyślenia i byłyby zbieżne z wysiłkami już podjętymi przez ECFA [2]) i CERN [3]), jak również inne europejskie i amerykańskie instytucje i organizacje.

---

[1]) LEP — ang. Large Electron-Positron Collider
[2]) ECFA — ang. European Collaboration on Future Accelerators
[3]) CERN — ang. European Organization on Nuclear Research

* * *

Pierwsza wersja tego artykułu została przedstawiona na konferencji „Zastosowanie Komputerów w projektowaniu i eksploatacji akceleratorów" 20—23 września 1983 r., w Berlinie Zachodnim.

Jest on poświęcony dziesiątej rocznicy utworzenia i rozpoczęcia komputerowej działalności Środowiskowego Centrum Obliczeniowego CYFRONET byłego Instytutu Badań Jądrowych w Otwocku-Świerku. Z tej okazji chciałbym podziękować bardzo serdecznie wszystkim moim współpracownikom i przyjaciołom za ich aktywne uczestnictwo i wkład w koncepcyjną, organizatorską, naukową i techniczną działalność Centrum [4]).

Pragnąłbym szczególnie podziękować mgr. Piotrowi Strzałkowskiemu za pomoc w ostatecznym ustaleniu kształtu tego artykułu, jak również za współpracę przy zagadnieniach związanych z systemem PSL/PSA. Bardzo dziękuję również jemu oraz mgr. inż. Jerzemu Grabowskiemu i mgr. Tomaszowi Czosnyce za współpracę przy sformułowaniu koncepcji tzw. systemów informatycznych eksperymentów.

---

[4]) Prof. R. Żelazny został odwołany ze stanowiska Dyrektora ŚCO CYFRONET w dniu 18 lipca 1983 r. (p. INFORMATYKA, nr 10/1983, str. 24)

LITERATURA

[1] Alford M. W.: A requirements engineering methodology for realtime processing requirements, IEEE trans. on Software Eng., vol. SE-3, 60, 1977

[2] Alford M. W., Davis C. G.: Experience with the software development system. W: Hünke H. (Ed): Software Engineering Environments. Horth-Holland, Amsterdam, New York, Oxford, 1981

[3] Bell T. E., Bixler D. C., Dyer M. E.: An extendable approach to computer-aided software requirements engineering. IEEE Trans. on Software Eng., vol. SE-3, 49, 1977

[4] Boyer R. S., Moore J. S.: A formal semantic of SRI Hierarchical Program Design Methodology. Computer Science Laboratory, SRI International, November 1978

[5] Boyer R. S., Moore J. S.: A Computational Logic. Academic Press, 1979

[6] Czosnyka T., Grabowski J., Strzałkowski P., Żelazny R.: Experiment Information Systems (a proposal), Preprint of the Institute of Nuclear Research, Otwock-Świerk, October 1981

[7] Dahl O. J., Hoare C.A.R.: Hierarchical program structures, W: Dahl O. J., Dijkstra E. W., Hoare C.A.R.: Structured programming Academic Press, 1972

[8] Davis C. G., Vick C. R.: The software development system. IEEE Trans. on Software Eng., vol. SE-3, 69, 1977

[9] Ehring H., Fey W.: A method for the specification of software systems: from formal requirements to algebraic design specifications, Preprint May 1982, Technical University Berlin, Institute für Software und Teoretische Informatik. Skrócona wersja tej pracy opublikowana jest w: Brauer W. (Ed), Informatik-Fachberichte 50, Springer Verlag, 1981

[10] Elliot W. D., Silverberg B. A., Levitt K. N.: A critique of HDM. Technical Report CSL-131, Computer Science Laboratory, SRI International, November 1981

[11] Gaulding S. N., Lawson J. D.: Process Desing Engineering — a methodology for real-time software requirements. Procedings 2nd International Software Engineering Conference, 1976

[12] Goldberg J.: Hierarchical System Development, Final Report, Computer Science Laboratory, SRI International, June 1978

[13] Guttag J. V., Horning J. J.: The algebraic specification of abstract data types. Acta Informatica, vol. 10, 27, 1978

[14] Guttag J. V.: Abstract data types and the development of data structures, CACM, vol. 20, 396, 1977

[15] Hamilton M., Zeldin S.: Higher Order Software — a methodology for defining software. IEEE Trans. on Softw. Eng. vol. SE-2, 9, 1976

[16] Hamilton M., Zeldin S.: The relationship between design and verification. The Journal of Systems and Software, vol. 1, 29, 1979

[17] Hamilton M., Zeldin S.: The functional life cycle model and its automation: USE. IT. Technical Report No. 36, Higher Order Software, Inc. Cambridge, MA, December 28, 1982

[18] Heacox H. C.: RDL: A Language for Software Development, ACM Sigplan Notices, vol. 14, 71, 1979

[19] Kopang R.: Process Design System — An integrated set of software development tools. Proceedings 2nd International Software Engineering Conference, 1976

[20] Liskow B.: Modular program construction using abstractions. W: Björner D. (Ed): Abstract Specifications. Lecture Notes in Computer Science, No. 86, Springer, Berlin, Heidelberg, New York, 1980
[21] Liskow B., Zilles S.: Specification techniques for data abstractions, IEEE Trans. on Software Eng., vol. SE-1, 7, 1975
[22] Parnas D. L.: A technique for software module specification with examples. CACM, vol. 15, 330, 1972
[23] Parnas D. L.: On the criteria to be used in decomposing systems into modules. CACM, vol. 15, 1053, 1972
[24] Robinson L., Levit K. N.: Proof technigues for hierarchically structured programs, CACM, vol. 20, 271, 1971
[25] Schneider H. J.: Techniques and formal tools for design, realization and evaluation of evolutionary information systems. Interner CIS Bericht 8/81, Technical University Berlin, Institute für Angewandte Informatik, 1981. Będzie opublikowane w: Hawgood I. (Ed), Proc. IFIP TC-3 Working Conference on Evolutionary Information Systems, North-Holland, Amsterdam, New York, Oxford
[26] Silverberg B. A.: On overview of the SRI Hierarchical Development Methodology. W: Hünke H. (Ed): Software Engineering Environments. North-Holland, Amsterdam, New York, Oxford, 1981
[27] Wasserman A. I., Gutz S.: The future of programming. CACM, vol. 25, 196, 1982
[28] Willis R. R.: AIDES Computer-aided design of software systems-II. W: Hunke H. (Ed): Software Engineering Environment. North-Holland, Amsterdam, New York, Oxford, 1981
[29] Wingrad T.: Beyond programming languages. CACM, vol. 22, 391, 1979
[30] Zilles S. N.: Algebraic specification of data types. Project MAC Progress Report, MIT, Cambridge, 1974.

WŁADYSŁAW UDRYCKI
WIESŁAW WILCZYŃSKI
Instytut Telekomunikacji
Politechnika Warszawska

CHILL — język programowania systemów komutacyjnych (3)

Instrukcje i struktury danych

Najprostszy program w języku CHILL rozpoczyna się od słowa **MODULE**, po którym następuje ciąg instrukcji zakończony słowem **END**. Ciąg ten zawiera instrukcje działań (ang. action statements) podejmowanych w programie, uzupełnione definicjami (ang. definition statements) i deklaracjami (ang. declaration statements) danych. Dane (ang. data objects), które mogą występować w postaci wartości (ang. value) lub zmiennych (ang. location [1]), są obiektami poddawanymi działaniom (ang. actions) zapisanym w programie.

WYBRANE INSTRUKCJE

Ciągi instrukcji (deklaracji i definicji), są objęte konstrukcjami nawiasowymi (ang. bracketed constructs) tworzącymi podaną blokową strukturę programu. Stosowanie instrukcji **IF**, pojedynczych lub zagnieżdżonych, pozwala na rozgałęzianie i selektywne wykonywanie wydzielonych segmentów. Jednakże, ze względu na czytelność programu, powinno się unikać głębokiego zagnieżdżania. Stopień zagnieżdżania zadań programu można zmniejszyć używając instrukcji **ELSIF**, np.:

```
DIGITS:
IF DIGIT = 0 AND DIGIT <10
  THEN VALUE := VALUE + DIGIT;
  ELSIF DIGIT = -1
   THEN NEG := TRUE;
   ELSE FIN := NOT FIRST;
FI DIGITS;
```

Dr inż. WŁADYSŁAW UDRYCKI jest absolwentem Wydziału Elektroniki Politechniki Warszawskiej. Studia (kierunek: telekomunikacja) ukończył w 1969 roku. W latach 1970—1974 odbył studia doktoranckie. Od 1975 roku jest pracownikiem Instytutu Telekomunikacji Politechniki Warszawskiej. Interesuje się zagadnieniami związanymi z językami i budową oprogramowania systemów komutacyjnych.

Życiorys WIESŁAWA WILCZYŃSKIEGO przedstawimy w numerze 1/85.

Dwie sprzężone ze sobą instrukcje warunkowe nie są zagnieżdżone. Przykład ten można zapisać bez użycia instrukcji **ELSIF**, wówczas sekwencja słów **EISE IF** rozpoczynałaby drugi warunek **IF** zagnieżdżony w poprzednim. Zapis bez zagnieżdżania zdań programu umożliwia także instrukcja **CASE**, np.:

```
CASE DIGIT OF
  (0 : 9): VALUE := VALUE + DIGIT;
  (-1): NEG := TRUE;
  ELSE FIN := NOT FIRST;
ESAC;
```

Trójścieżkowe rozgałęzienie zależy od wartości zmiennej DIGIT nazywanej selektorem wyboru (ang. case selector). Konstrukcję w nawiasach nazywa się etykietą wyboru (ang. case label).

Bardziej złożone warunki wygodniej jest przedstawić w postaci tzw. tablicy decyzyjnej (ang. decision table), np.:

```
CASE I,        C,          B,          OF
     (1),      ('A')       (TRUE):     X := 1;
     (2 : 5),  ('D' : 'F'), (FALSE):   X := 2;
     (ELSE),   ('G' : 'Z'), (*):       X := 3;
     ELSE
ESAC;
```

Wartość zmiennej X podstawia się w zależności od zestawu wartości trzech obiektów: zmiennej całkowitej I, zmiennej znakowej C i zmiennej logicznej B, przy czym **(EISE)** oznacza dowolną wartość poza wymienionymi w pozostałych alternatywach, a apostrof (*) — wartość nieokreśloną.

Instrukcje pętli (ang. loop statements) umożliwiają kontrolowane powtarzanie wydzielonej części programu — zwanej ciałem pętli (ang. loop body) — zamkniętej nawiasami **DO** i **OD**. Bezpośrednio po słowie **DO** następuje część sterująca (ang. control part) pętli. W zależności od sposobu sterowania wyróżnia się dwa rodzaje pętli: sterowane warunkiem **WHILE** i interowane **FOR**, które mogą występować łącznie, np.:

[1]) W CHILLU operuje się specyficzną terminologią, w której zmienne nazywane są locations — miejsca, lokacje. Według podanej w opisie CHILLA definicji są one abstrakcyjnymi pojemnikami mogącymi przechowywać wartości. Tak pojmowany synonim zmiennej jest bardzo ilustracyjnym pojęciem.

```
DO FOR CH DOWN IN CHAR ('A' : 'Z')
   WHILE CH / = 'F';
   OUTCHAR (CH);
OD;
```

Wskutek wykonania tej pętli, na urządzenie wyjściowe zostaną wyprowadzone znaki od Z do G, w kolejności odwrotnej do alfabetycznej, co oznaczone jest słowem **DOWN.**

Dzięki instrukcjom pętli można w łatwy i niezawodny sposób przeglądać tablice (ang. arrays scanning). Jeżeli po słowie **IN** następuje nazwa tablicy, to licznik pętli wskazuje kolejne elementy tablicy, np.:

```
DO FOR ELEMENT IN A;
   SUM1 + : = ELEMENT;
   SUM2 + : = ELEMENT * ELEMENT;
OD;
```

Wskutek wykonania powyższej pętli obliczone zostaną dwie sumy — elementów i kwadratów elementów tablicy A. Warto przy tym zwrócić uwagę na łączny zapis operacji, sumowania i przypisywania np. pierwsza instrukcja wewnątrz pętli jest równoważna zapisowi:

SUM1: = SUM1 + ELEMENT;

Powyższa konstrukcja, pozwalająca na przeglądanie tablic bez jawnego indeksowania, stanowi ważny środek zwiększania niezawodności oprogramowania. Jej zaletą jest także to, że zmiana deklaracji tablicy nie pociąga konieczności zmiany w części sterującej pętli.

Przy przeglądaniu tablic można zmienić kierunek stosując słowo **DOWN**, można również użyć kilku liczników, np.:

```
DO FOR ELEMENT_A IN A,
       ELEMENT_B IN B;
   SUM + : = ELEMENT_A + ELEMENT B;
OD;
```

Przeglądane tablice nie muszą być równoliczne — pętla przestaje się wykonywać, gdy choć jeden z jej liczników wskazuje koniec tablicy.

Ważną postacią pętli FOR jest pętla nieskończona (ang. indefinite loop), zawierająca w części sterującej słowo **EVER**, np.:

```
DO FOR EVER;
   I + : = 1 ;
   WAIT (5);
OD;
```

Wartość zmiennej I jest zwiększana bez końca co pięć sekund. WAIT jest zdefiniowaną gdzie indziej procedurą, która na określony czas wstrzymuje wykonywanie programu.

W przypadku pętli nieskończonych, jak również innych instrukcji złożonych, niezbędne jest zapewnienie możliwości przerywania ich wykonywania. Może to następować zarówno w przypadku poprawnego zakończenia działań zapisanych w ciele instrukcji, jak i w przypadku wystąpienia sytuacji błędnej. Obsługę obydwu sytuacji umożliwia instrukcja **EXIT**, która — oprócz słowa kluczowego — zawiera nazwę etykiety identyfikującej konstrukcję nawiasową lub instrukcję złożoną, np.:

```
1. CONSUMER:
2. DO FOR EVER;
3.    EMPTY_BUFFER (STATUS);
4    IF STATUS = ERROR
5.   THEN
6.      EXIT CONSUMER;
7.   FI;
8. RETURN_BUFFER ();
9. OD CONSUMER;
```

Jeżeli podczas wykonywania pętli nieskończonej zmienna STATUS przybierze wartość ERROR, to pętla zostanie przerwana instrukcją EXIT w linii 6, a wykonywanie programu będzie kontynuowane po linii 9.

W CHILLU istnieje także instrukcja **GOTO** zalecana jednak do stosowania tylko w wyjątkowych przypadkach, ponieważ jej użycie może łatwo spowodować naruszenie struktury programu.

STRUKTURY DANYCH

Każdy obiekt używany w programie jest opisany przez typ (ang. mode) lub klasę (ang. class), które określają zbiory możliwych wartości oraz operacje, jakie można na nich wykonywać. Opis właściwości obiektów jest więc uzupełnieniem opisu działań, co pozwala rozszerzyć analizę wewnętrznej niesprzeczności programu. Niepoprawne konstrukcje programowe są wykrywane już podczas statycznej analizy ich tekstu przez kontrolę zgodności typów. Dzięki temu błędy, które byłyby wykryte dopiero w trakcie wykonywania programu, mogą zostać usunięte we wcześniejszych etapach procesu programowania.

Ze względu na czytelność i niezawodność programu, korzystne jest, by właściwości obiektów były grupowane i opatrywane wspólną nazwą typu. Służą temu możliwości definicyjne CHILLA w postaci definicji nowego typu **NEWMODE** i tworzenia typów synonimowych **SYNMODE.**

Definiowanie nowych typów w CHILLU pozwala na hierarchiczne rozbudowywanie struktur danych z wykorzystaniem struktur prostszych. Właściwości nowo tworzonych typów są identyfikowane przez nazwę nadawaną przez programistę. Dzięki temu można upodobnić nazwy stosowane w programie do nazw pojęć używanych przy rozwiązywaniu problemu, co zilustrowano w poniższym przykładzie.

```
1.  NEWMODE JUNCTIONS = INT,
2.              TIMES = INT,
3.              CALL_RECORDS = STRUCT
                               (JUNCTION_NO  JUNCTIONS,
4.                              ANSWER_TIME  TIMES,
5.                              STANDARD_RATE  BOOL) ;
6.  DCL  CURRENT_CALL_RECORD  CALL_RECORDS;
7.  CURRENT_CALL_RECORD.JUNCTION_NO : = 1375;
8.  CURRENT_CALL_RECORD.ANSWER_TIME : = 3;
9.  CURRENT_CALL_RECORD.JUNCTION_NO : =
10.   CURRENT_CALL_RECORD.ANSWER_TIME;  (*  niezgodność
                                              typu *)
```

W liniach 1 i 2 zdefiniowane są nowe typy JUNCTIONS i TIMES. Mimo że obydwa mają właściwości typu **INT,** to w rozumieniu reguł zgodności typów są one różne od siebie i od każdego innego typu. W następnej linii CALL_RECORDS zdefiniowany jest jako typ o postaci struktury z trzema polami, z których dwa to uprzednio zdefiniowane typy. W linii 6 zadeklarowana jest zmienna CURRENT_CALL_RECORD typu CALL_RECORDS, której pola JUNCTION_NO i ANSWER_TIME są inicjowane liczbami całkowitymi w liniach 7 i 8. Natomiast przypisanie w liniach 9 i 10 nie jest poprawne, gdyż obydwa pola są różnych typów. Jest oczywiste, że liczba łączy i czas odpowiedzi (ang. junction number, answer time) są znaczeniowo różnymi wielkościami, mimo że obydwie opisuje się liczbami całkowitymi.

Inny rodzaj typów definiowanych, typ synonimowy, pozwala na nadawanie nowych nazw tym samym typom bez zmiany semantyki programu, np.:

```
SYNMODE SIGNAL_MATRICES = ARRAY(1 : 3) ARRAY(1 : 5) INT;
DCL MATRIX_A SIGNAL_MATRICES;
 MATRIX_B = ARRAY(1 : 3)  ARRAY(1 : 5)  INT;
MATRIX_A : =  MATRIX_B;          (* zgodność typów *)
```

Tablica MATRIX-A jest zadeklarowana przy użyciu typu synonimowego SIGNAL_MATRICES zdefinowanego jako dwuwymiarowa tablica typu **INT** o dokładnie takich samych rozmiarach jak deklarowana zmienna MATRIX_B. Zatem przypisanie w ostatniej linii nie narusza reguł zgodności typów.

Jeśli istnieje możliwość wyboru między **NEWMODE** i **SYNMODE,** to zaleca się stosowanie **NEWMODE** jako środka skuteczniejszego z uwagi na kontrolę zgodności typów.

Wygodnym środkiem zwiększania czytelności programu i jego parametryzacji są synonimy stałych. Definicja synonimu stałej tworzy nowy literał stałej, który może być opatrzony nazwą zrozumiałą dla użytkownika. Czyni to program bardziej czytelnym i ułatwia wprowadzanie zmian, np.:

```
SYN BUFFER_SIZE = 80;
DCL BUFFER_POOL  ARRAY(1 : BUFFER_SIZE)  INT;
```

Zdefiniowany synonim stałej o nazwie BUFFER_SIZE może być użyty w innym miejscu programu, np. do określenia rozmiaru i parametryzacji tablicy BUFFER_POOL.

dokończenie na str. 21

Kontakt z mikroinformatyką można zacząć różnie. Niektórzy znacznym nakładem finansowym sprowadzają z zagranicy gotowy mikrokomputer. Inni składają go sami, ponosząc wprawdzie mniejsze koszty, ale borykając się z większymi trudnościami. MINI-MONITOR jest propozycją właśnie dla tych drugich. Zawiera on stosunkowo niewiele funkcji, ale zajmuje niewiele ponad 800 bajtów. Zapisany w pamięci EPROM (wystarczy tu jeden układ 2708), może oddać nieocenione usługi przy oprogramowywaniu własnej konstrukcji.

MINI-MONITOR wymaga dołączenia dwóch procedur — wprowadzania i wyprowadzania znaku na konsolę operatorską. Są one ściśle zależne od zastosowanej konstrukcji i przy braku standardowych rozwiązań nie sposób przewidzieć wszystkich możliwych opcji. Warto więc podkreślić, że autor przygotował MINI-MONITOR tak, aby można go było wykorzystać w dowolnej konstrukcji.

MINI-MONITOR dla 8080 i Z80

Przedstawiony poniżej program realizuje pięć funkcji. Umożliwia przeglądanie i zmianę zawartości pamięci oraz pozwala na uruchamianie programów w języku maszynowym mikroprocesorów 8080 i Z80.

MINI-MONITOR umieszczony jest w pamięci od adresu 0. Umożliwia to rozpoczęcie realizacji programu przez podanie sygnału zerującego RESET. Pojawienie się na konsoli operatora znaku ">" oznacza gotowość programu do pracy (wykonywania poleceń). Polecenie składa się z kodu polecenia (litera: M, H, A, G lub B), jednego lub dwóch argumentów i znaku CR (powrót karetki, w kodzie ASCII--0DH). Argumenty podaje się w kodzie szesnastkowym. W przypadku podawania dwóch argumentów, należy je rozdzielić co najmniej jedną spacją (odstępem).

Polecenie:

M — przeglądanie i zmiana zawartości pamięci.
M adres [CR]

Na konsolę wyprowadzony zostaje adres i zawartość komórki pamięci o podanym adresie. Program oczekuje teraz na wprowadzenie znaku (znaków). Po wprowadzeniu kodu CR, program powtarza działanie dla kolejnej komórki pamięci (p. przykład 1). Aby zmienić zawartość komórki, należy wprowadzić dwie cyfry szesnastkowe, a następnie CR (p. przykład 2).

Wprowadzenie kropki "." kończy wykonywanie polecenia.

H — wyprowadzenie zawartości obszaru pamięci (szesnastkowo) na konsolę operatora.
H adres1 adres2 [CR]

Zawartość obszaru pamięci zostaje wyprowadzona z przestrzeni pomiędzy adres1 ... adres 2 w wierszach po osiem bajtów w każdym (p. przykład 3).

A — wyprowadzenie zawartości obszaru pamięci (znaki ASCII)
A adres1 adres2 [CR]
Jak wyżej. Wartości spoza tablicy kodów ASCII reprezentuje kropka "." (p. przykład 4).

G — rozpoczęcie realizacji programu
G adres [CR]
Rozpoczyna wykonywanie programu użytkownika od podanego adresu.

B — ustawienie lub usunięcie pułapki
B adres [CR]
Do pamięci, pod wskazanym adresem, wstawiona zostaje instrukcja CALL BR. Napotkanie tej instrukcji w czasie wykonywania programu powoduje wyprowadzenie zawartości rejestrów w postaci: R W1 (W2). R oznacza tu nazwę pary rejestrów (AS to odpowiednik PSW w 8080, ST to dwa bajty z wierzchołka stosu, czyli adres instrukcji po CALL BR), W1 — zawartość rejestrów, a W2 — zawartość komórki pamięci wskazanej przez adres znajdujący się w parze rejestrów (p. przykłady 5 i 6).
B [CR]

Wstawia na miejsce CALL BR poprzednią zawartość (p. przykład 7). Nie

Przykład 1. Przeglądanie zawartości pamięci

```
>M1000[CR]
1000 06[CR]
1001 41[CR]
1002 3E[CR]
1003 42[CR]
1004 21[CR]
1005 01[CR]
1006 44 .[CR]
>
```

Przykład 2. Zmiana zawartości komórek pamięci

```
>M1000[CR]
1000 20 06[CR]
1001 46 41[CR]
1002 49 3E[CR]
1003 4C 42[CR]
1004 45 21[CR]
1005 20 01[CR]
1006 52 44[CR]
1007 45 C3[CR]
1008 41 00[CR]
1009 44 20[CR]
100A 20 .[CR]
>
```

Komputerku, jesteś wolny!

Prezes Głównego Urzędu Ceł do profesora Iwo Białynickiego-Biruli (Zakład Fizyki Teoretycznej PAN):

W związku z pismem z 7 listopada 1981, proponującym zmianę zasad przywozu z zagranicy komputerów „osobistych", oraz nawiązując do poprzedniej korespondencji w tej sprawie, uprzejmie informuję Obywatela Profesora, że poleciłem urzędom celnym wydawanie „z urzędu", tzn. w trakcie odprawy celnej i bez żądania wniosków zainteresowanych, pozwoleń na przywóz z zagranicy komputerów „osobistych" i „domowych".

Przyjęte rozwiązanie jest jedynym możliwym w obowiązującym stanie prawnym, gdyż objęcie ewentualnie komputerów poz. 58 pkt 4 taryfy celnej przywozowej nie jest możliwe ze względów formalnych (Systematyczny Wykaz Wyrobów, na którym oparta jest nomenklatura taryfy celnej, zalicza wszelki sprzęt komputerowy do branży 092).

Polecone zasady będą obowiązywały przejściowo do czasu zmiany rozporządzenia Ministra Handlu Zagranicznego ustanawiającego taryfę celną.

Jakież to niewiarygodne! Jeszcze niedawno powoływano się w Urzędzie Ceł na dobro państwa — zmuszonego bronić się przed komputerami o niezbadanych możliwościach... A tu naraz: proszę, droga wolna. I — co więcej — jest to droga najwygodniejsza; cło pozostaje bowiem na dotychczasowym, niskim poziomie. Doprawdy, zadziwiające zwycięstwo rozsądku.

Tak czy inaczej, wydarzenie jest szczęśliwe i nie ma co kręcić głową. Rozważmy raczej możliwe następstwa. Skoro procedura celna została uproszczona, spodziewać się teraz można znacznego dopływu sprzętu. Nie tylko przywożonego przez Polaków z zagranicy, ale też przysyłanego stamtąd. Być może nadawcami będą także organizacje (np. polonijne), dla których rozwój cywilizacyjny Polski nie jest sprawą bez znaczenia. Pomyślmy więc — jeśli zdarzy się tak, w jaki sposób zdołamy to przyjąć. Kto i na jakiej zasadzie? Czy nie jest to dobry moment na dokonanie dalej idącej samoorganizacji amatorów mikroinformatyki.

Zresztą, na zorganizowanie się samemu każdy moment jest najlepszy.

ZBIGNIEW GLUZA

Przykład 3. Wyprowadzenie zawartości pamięci (szesnastkowo)

```
>H1000 100B[CR]
1000 06 41 3E 42 21 01 44 C3
1008 00 20 20 43 44 45 46 47
>
```

Przykład 4. Wyprowadzenie zawartości pamięci (znaki ASCII)

```
>A1000 100B[CR]
1000 . A>B ! . D .
1008 . C D E F G
>
```

Przykład 5. Ustawienie pułapki (punktu wstrzymania realizacji programu)

```
>B1004[CR]
>M1004[CR]
1004 CD[CR]
1005 16[CR]
1006 01 .[CR]
```

Przykład 6. Rozpoczęcie realizacji programu i wyprowadzenie zawartości rejestrów po wstrzymaniu spowodowanym napotkaniem pułapki

```
>G1000[CR]
AS 4200 (C1)
BC 410F (45)
DE 1000 (06)
HL 03AE (10)
ST 1007 (C3)
```

Przykład 7. Usunięcie pułapki

```
>B[CR]
>M1004[CR]
1004 21[CR]
1005 01[CR]
1006 44[CR]
1007 C3 .[CR]
```

można ustawić jednocześnie więcej niż jednej pułapki.

•

MINI-MONITOR może pracować na dowolnym mikrokomputerze wykorzystującym mikroprocesor 8080 lub Z80.

Kod programu zajmuje stałe miejsce w pamięci (ROM lub RAM) od adresu 0 do 29FH. (Kolejne bajty podane na wydruku należy umieścić w pamięci poczynając od adresu 0). Program wymaga 48 bajtów pamięci RAM; 24 bajty — obszar roboczy, 24 — stos. Początkowy adres obszaru roboczego należy podać w komórkach: 29DH — bajt mniej znaczący adresu i 29EH — bajt bardziej znaczący. Adres ten będzie jednocześnie wykorzystany przy ustaleniu wskaźnika stosu.

Kod MINI-MONITORA wymaga uzupełnienia o procedurę wprowadzania znaku z konsoli (klawiatury) do akumulatora — CZYT oraz procedurę wyprowadzania znaku z akumulatora na konsolę (ekran monitora) — **PISZ.** Procedury takie zależą od konstrukcji mikrokomputera. Adresy procedur należy umieścić w programie pod adresami podanymi w tabeli:

Na wydruku przedstawiono kod programu MINI-MONITOR w postaci

Procedura	Bajt mniej znaczący	Bajt bardziej znaczący
PISZ	28CH	28DH
CZYT	297H	294H

```
A)type mintmon.hex
:100000002A9D02F90610AF772305C20600CD08022F
:10001000E3ECD8702CD45022A9D02EB214500AF31
:10002000AFE0DCA0D008ECA3600EFFBECA0D0044
:10003000232323C32000E5CDC001E1235E23562103
:10004000D00E5D5C94D8F00475500485E0042C8F8
:10005000004156B1FFAFCD6B025E2356D5C9AFCD2F
:10006000B02CD7F02CD0802CD17020608CD120229
:10007000ECDF1012303C26D00E5AFCD6902CD7FD4
:10008000020EBE1E5CD7502E1D8CD0802C36800AF0F
:10009000CD6B02CD7F02CD0802CD1702CD12027EBC
:1000A000CDF101CD1202E5CD4502AF2A9D02EBE173
:1000B0001AFE0DC2BA0023C39600FE2ECBE5CD9CE1
:1000C000017CE17723C39600AF2A9D02237EFE0DBB
:1000D000CAFF00AFCD6502E5AFCD6B025E2356EBE4
:1000E000D1E506037E12231305C2E400AFCD6702FB
:1000F000D1732372EB36CD231117017323720C9AF6D
:10010000CD67025E2356D5AFCD6502D106037E12C0
:10011000231305C20E01C9E5D5C5F5CD0802E106D8
:1001200041 0E53CD2002CD2B02E106420E43CD20DD
:10013000020D2B02E106440E45CD2002CD2B02E17B
:10014000064B0E4CCD2002CD2B02E106530E54CDB5
:10015000202CD2B02C9AFCD6B02CD7F02CD0B02AC
:10016000CD17020608CD1202AF7EDE203E2EDA7DCC
:10017000017EDE7ADA7C013E2EC37D017ECDB702D0
:1001B00023AF05C26501E5AFCD6902CD7F02EBE1BA
:10019000E5CD7502E1DBCD0B02C36001F5C52100A7
:1001A0000006041AD630FE0AFAAD01D607E60FB5EE
:1001B0006F1305CABD0129292929C3A301C1F1C9AA
:1001C0002A9D0223EBCDE801CD9C01D5E5AFCD6B97
:1001D00002D1732372D1CDE801CD9C01D5E5AFCD1D
:1001E0006902D1732372D1C9AF1AFE20C013C3E8CC
:1001F00001F51F1F1F1FCDFA01F1E60FC630FE3AB1
:1020000DA0502C607C3B7023E0DCDB7023E0AC348
:10021000B7023E20C3B7027CCDF1017DCDF101C96B
:1002200078CDB70279CDB702C31202CD1702CD1295
:10023000023E28CDB7027ECDF1013E29CDB702CD39
:1002400012020C30B022A9D02AFCD9302CDB7027726
:1002500023FE0BC25F022B36202B3620C34B02FE45
:1002600000DC8C34B02C602C602C602C60F4F06002A
:100270002A9D0206C9237CB537CB7B957A9CC97E23
:10028000F5237E67F16FC9F5E5D5C5CD0000C1D175
:0F029000E1F1C9E5D5C5CD0000C1D1E1C900003C
:0000000000
```

Wydruk kodu programu MINI-MONITOR w formacie INTEL-HEX (ostatni bajt w wierszu jest bajtem kontrolnym).

szesnastkowej. Wersję źródłową w języku ASSEMBLER 8080 można otrzymać w firmie CSK.

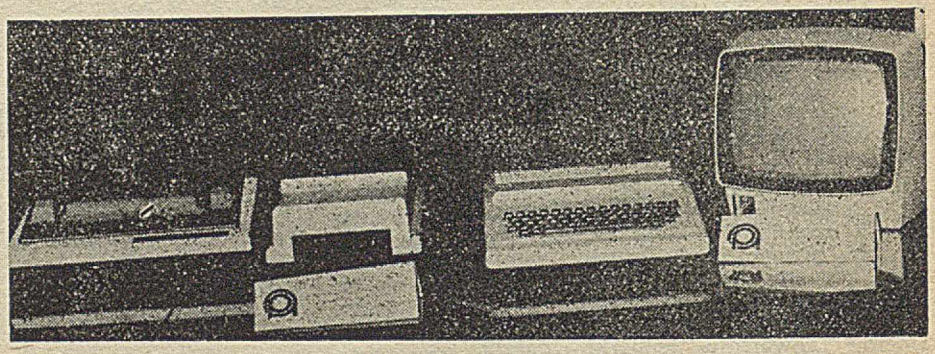
JERZY DWORZECKI
Computer Studio Kajkowski
Gdynia

Zgodnie z obietnicą, zamieszczamy opis kolejnego mikrokomputera produkowanego w kraju. W porównaniu do ZX81 jest to niewątpliwie rozwiązanie o klasę lepsze. Niestety, również sporo droższe.

AMEPROD ocenia możliwości produkcyjne na około trzysta egzemplarzy rocznie. Nie trudno więc wyrokować, że okaże się to kroplą w morzu potrzeb. Tym bardziej, że krążą wieści o podejmowanej przez PKP próbie zakupienia... tysiąca egzemplarzy. W krajach zachodnich duże zamówienie to olbrzymia szansa dla producenta. U nas może oznaczać... kilkuletnie zniknięcie firmy z rynku. Przyszłość pokaże, czy AMEPRODOWI uda się pogodzić ambicje upowszechniania mikroinformatyki z „propozycjami nie do odrzucenia".

Komputer osobisty AC 805

Doświadczenie zebrane przez ponad dwa lata produkcji i promocji komputera osobistego ZX81 skłoniły Przedsiębiorstwo Zagraniczne AMEPROD do podjęcia prac nad własnymi systema-



mi komputerowymi. Prezentowany komputer **AC 805**, którego pierwszy pokaz odbył się na MTP w 1984 roku, jest wersją podstawową w serii **AC 800**, w skład której wchodzą ponadto modele: **AC 815** oraz **AC 825.**

W zwartej obudowie, o ładnym, opływowym kształcie, umieszczono płytę CPU, płytę klawiatury, a także zasilacz dostarczający niezbędnych napięć dla systemu. Rolę jednostki centralnej spełnia 8-bitowy mikroprocesor Z80. Minimalny system obliczeniowy składa się z AC 805, monitora ekranowego lub telewizora (pracującego jako monitor ekranowy) i magnetofonu kasetowego (pamięć zewnętrzna). AC 805 został wyposażony w **Mikro System,** interpreter języka **BASIC** (zgodny ze standardem MICROSOFT, zbliżony do wersji zastosowanej w NASCOM COMP.) oraz **edytor ekranowy.** Wymienione oprogramowanie systemowe rezyduje w pamięci stałej ROM, o pojemności 12 KB. Przewidziano trzy wersje pamięci dynamicznej RAM o pojemnościach: 16, 32 oraz 48 KB.

Uruchomienie AC 805 jest bardzo proste — sprowadza się do połączenia go z monitorem i magnetofonem oraz włączenia do sieci. Po włączeniu zgłasza się Mikro System. Użytkownik ma do dyspozycji 19 dyrektyw, które umożliwiają:

— wejście do interpretera BASICA — „zimne"

MIKROKOMPUTER AC 805

jednostka centralna: Z80
oprogramowanie:
- interpreter języka BASIC według standardu MICROSOFT (8 KB)
- MONITOR o nazwie Mikro System
- opcjonalnie system operacyjny kompatybilny z CP/M 2.2

pamięć ROM — 12 KB
pamięć RAM — 16, 32 lub 48 KB
pamięć masowa:
- standardowo magnetofon (modulacja FSK 300 lub 1200 bodów)
- opcjonalnie dyski elastyczne 5,25″ — 1 MB
- opcjonalnie twarde dyski (WINCHESTER) — 10 lub 30 MB

wyświetlanie: 24 linie po 32 znaki, 96 znaków ASCII, 32 znaki graficzne, 128 znaków programowalnych, możliwa pseudografika
sprzęgi: szeregowy — 2 × RS 232C, 150... 4800 bodów; równoległy — 24 linie TTL
klawiatura: QWERTY; mata gumowa, plastikowe klawisze
dokładność obliczeń: 7 cyfr
cena: model podstawowy — 400 tys. zł (sierpień 1984)
producent: AMEPROD, 61-632 Poznań, Kmieca 20A, tel. 22 18 79
termin realizacji zamówienia: zależy od liczby zamówień, produkcja w 1985 roku — ok. 300 egzemplarzy

— wejście do interpretera BASICA — „gorące"
— zmianę konfiguracji wejść i wyjść
— rozpoczęcie realizacji programu w kodzie maszynowym
— zapis i odczyt programów przy współpracy z magnetofonem
— zmianę szybkości transmisji portów szeregowych
— ustawienie punktu wstrzymania (pułapki) dla wykonywanego programu
— kontrolę i aktualizację zawartości poszczególnych lokacji pamięci i rejestrów.

AC 805 został zaprojektowany jako komputer osobisty przeznaczony do zastosowań profesjonalnych, który z powodzeniem może pełnić rolę pomocy dydaktycznej lub narzędzia wspomagającego inżyniera. Z tego powodu szczególnie starannie opracowano komunikację komputera z urządzeniami zewnętrznymi. AC 805 ma dwa porty szeregowe (w standardzie RS 232 C) AUX i HOST, o ustawianej prędkości transmisji w zakresie 150—4800 bodów, oraz 24-bitowy programowalny port równoległy. Transmisja szeregowa może odbywać się bez udziału jednostki centralnej, dzięki czemu istnieje możliwość łączenia komputerów AC 850 ze sobą, przy jednoczesnej transmisji pomiędzy komputerem a dołączonym do portu szeregowego urządzeniem zewnętrznym. Port równoległy umożliwia dołączenie do systemu nietypowych urządzeń peryferyjnych, np. rejestratorów, sterowników czy drukarek i ploterów współpracujących poprzez złącze równoległe.

Klawiatura w układzie typu QWERTY została zbudowana z maty gumowej i nakładek z tworzywa sztucznego. Zastosowanie maty z histerezą mechaniczną zapewnia użytkownikowi jednoznaczną informację o naciśnięciu klawisza. Klawisze specjalne (np. CLEAR, HOME) znacznie ułatwiają pisanie programów i ich redagowanie. Współpraca z monitorem odbywa się poprzez wyjście video lub modulator UHF, pracujący na 36 kanale. Układ obsługi monitora ekranowego wyświetla 24 wiersze po 32 znaki i działa na zasadzie pamięci obrazowej (ang. video-RAM). Pamięć obrazowa jest adresowana jak pamięć mikrokomputera.

Mikrokomputer wyposażony jest w dwa generatory znaków. Generator podstawowy zawiera pełen zestaw znaków kodu ASCII, z dużymi i małymi literami oraz kilkunastoma znakami graficznymi. Rezyduje on w pamięci ROM. Drugi — dodatkowy generator — pozwala na zdefiniowanie 128 znaków przez użytkownika. Dzięki temu możliwe jest uzyskanie pseudografiki w całym obszarze obrazu, a także tworzenie nietypowych znaków alfabetu lub też drugiego, alternatywnego alfabetu (np. cyrylicy). Dostęp do pamięci obrazowej nie jest zsynchronizowany z sygnałami wygaszania plamki [1]).

Podkreślając walory dydaktyczne systemu, należy zwrócić uwagę na możliwość uruchamiania programów (pod kontrolą Mikro Systemu) wprowadzonych w kodzie maszynowych ich redagowania oraz współpracy z programami napisanymi w języku BASIC.

Pisanie i uruchamianie programów w BASICU wydatnie ułatwia edytor ekranowy z kursorem poruszającym się po ekranie we wszystkich kierunkach — z możliwością usuwania i wstawiania linii oraz wymiany zawartości linii programu.

Unikalną cechą AC 805 jest możliwość przygotowania programów w kodzie maszynowym dla zestawu laboratoryjnego ZLA 01 [2]), a wynika to z przyjęcia takiego samego standardu zapisu na taśmie magnetofonowej w obu urządzeniach (modulacja FSK). Cecha ta zwiększa możliwości dydaktyczne AC 805, jako narzędzia wspomagającego laboratoryjną naukę techniki mikroprocesorowej.

Prezentowany komputer jest wersją podstawową z rodziny AC 800, w ramach której w roku 1985 planowane jest uruchomienie produkcji wersji **AC 825**, współpracującej z dyskiem typu **WINCHESTER-AC 925 WD.** Trwają też prace nad rozpoczęciem produkcji wersji AC 815, współpracującej ze stacjami dysków elastycznych 5,25″ — **AC 815 FDD.**

Pamięci zewnętrzne będą budowane w oparciu o napędy firmy BASF, a kontrolowane przez system operacyjny kompatybilny z CP/M 2.2. W wersji 815 użytkownik będzie miał do dyspozycji łącznie 1 MB pamięci masowej (na dwóch dyskach, przy zapisie dwustronnym). Dla modelu 825 planujemy zastosowanie dysków WINCHESTER 10 i 30 MB.

Nr programu	Czas realizacji
1	1,99
2	7,44
3	18,00
4	18,50
5	19,75
6	30,35
7	43,75
Suma	139,78

Wykonaliśmy dla AC 805 testy opublikowane w trzecim tegorocznym numerze INFORMATYKI (za PPC Computer Journal V1N1010, 1982). W tabeli podano czasy wykonywania poszczególnych programów: Suma punktów wskazuje, że AC 805 dorównuje szybkością działania podobnym komputerom tej klasy produkowanym na świecie.

RYSZARD STEFANOWSKI
AMEPROD

[1]) Konsekwencją tego może być zauważalne migotanie obrazu — w przypadkach gdy wymieniana jest znaczna część informacji zawartej w pamięci obrazu. Poza niektórymi bardzo specyficznymi programami, użytkownik nie spotyka się ze wspomnianym objawem (przypis AJP)

[2]) Zestaw laboratoryjny ZLA 01 jest jednopłytowym mikrokomputerem zbudowanym z wykorzystaniem procesora INTEL 8080 (MCY 7880) produkowanym przez P.Z. AMEPROD (przypis autora).

Tekst został opublikowany w porozumieniu z producentem, a także z finansowym wsparciem z jego jego strony. *EO/1483/K/84*

- Ponad 300 członków zrzesza węgierski klub mikro o nazwie HCC (od angielskiego: Homebrew Computer Club). W ramach klubu działa wiele sekcji zainteresowań (m.in. SINCLAIR, APPLE, VC20, RRS 80). Klub nie tylko pomaga swoim członkom w zdobywaniu wiedzy mikroinformatycznej, ale również — w zdobywaniu części zamiennych i układów.

Zamieszczając pierwszy tekst o jakimś mikrokomputerze patrzymy na niego z perspektywy potencjalnego nabywcy. Dopiero w następnych publikacjach można znaleźć bardziej szczegółowe informacje, adresowane już do użytkowników. TI 99/4A jeszcze nie opisywaliśmy. Ponieważ jednak nie będzie już nam w kraju przybywać użytkowników tego mikrokomputera (TEXAS INSTRUMENTS zaprzestał jego produkcji), publikujemy materiał użyteczny dla tych, którzy już go mają. Chociaż nie tylko dla nich. Wiele osób przymierza się do własnych 16-bitowych konstrukcji, ciułając z trudem zebrane „zielone". Być może przedstawione niżej kompromisowe rozwiązanie stanie się źródłem inspiracji — sprowadzającej konstrukcję do poziomu zasobności kieszeni. Nawet w 16-bitowych systemach pamięć jest bowiem najdroższym elementem.

Organizacja pamięci wewnętrznej mikrokomputera TI 99|4A

TI 99/4A jest zmodernizowaną wersją znanego już wcześniej mikrokomputera TI 99/4. Zasadnicza idea konstrukcji pozostała niezmieniona. Modyfikacje dotyczyły wzbogacenia mikrokomputera oraz ułatwienia jego obsługi. Również przy konstruowaniu uproszczonej wersji (TI 99/2) wykorzystano wiele rozwiązań stosowanych w poprzednich systemach.

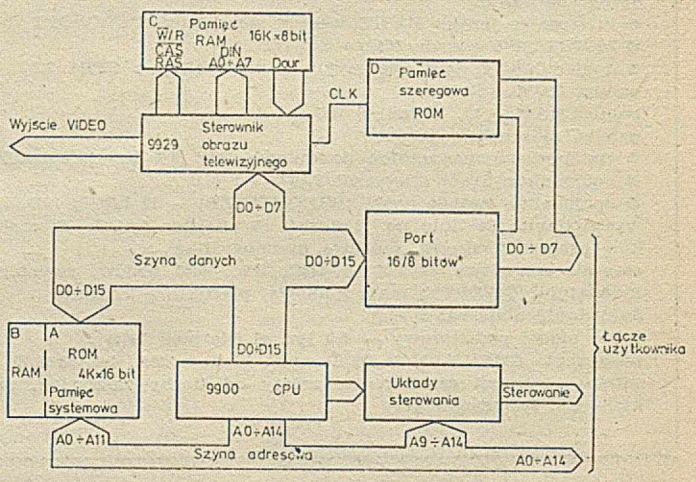
Wśród zalet systemu TI 99/4A szczególnie istotne wydają się następujące cechy:
— stosunkowo niska cena (ok. 150 dol.[1]))
— wbudowany interpreter języka BASIC (wersja TI BASIC z możliwością rozszerzenia) wzbogaconego o instrukcje graficzne, kontroli manipulatorów i sterowania dźwiękiem
— dobra klawiatura o standardowych rozmiarach i konstrukcji.
Analizując podzespoły wykorzystane w konstrukcji mikrokomputera, warto podkreślić fakt użycia procesora 16-bitowego, a więc mającego większe możliwości przetwarzania w porównaniu do procesorów o mniejszej długości słowa. Nietypowa struktura wewnętrzna procesora (organizacja typu pamięć-pamięć, bez rejestrów wewnętrznych) umożliwia łatwą zmianę kontekstu programowego i obsługę przerwań. Mimo że sam procesor został opracowany w 1976 roku, to jego parametry nie odbiegają od parametrów procesorów obecnie produkowanych (czas wykonania instrukcji dodawania — 4,7 μs dla częstotliwości zegara 3MHz). Do jego zalet należy zaliczyć osobne wyprowadzenie szyny danych i szyny adresowej oraz stosunkowo dużą liczbę linii sterujących.

Przy potencjalnie dużych możliwościach systemu, zaskakuje jednak powolność realizacji programów obliczeniowych.

Ewentualna rozbudowa systemu jest zasadniczo możliwa tylko w oparciu o moduły i urządzenia dostarczane przez producenta. Wiąże się to z niestandardowym rozwiązaniem złączy systemu oraz zastrzeżeniem producenta, iż współpraca z zewnętrznymi urządzeniami jest możliwa tylko po dołączeniu modułów programowych (w postaci pamięci ROM), współpracujących z programem nadzorującym pracę systemu. Oczywiście, użycie modułów produkowanych przez firmę TEXAS INSTRUMENTS jest rozwiązaniem najprostszym i optymalnym z punktu widzenia poprawności współpracy wszystkich urządzeń. Jednak mają one poważną wadę — koszt wielu modułów jest większy od kosztu mikrokomputera w wersji podstawowej (np. przystawka sterująca RS232C — ok. 200 dol., sterownik dysków elastycznych — ok. 220 dol.). Zapewne większość hobbystów będzie starała się rozbudowywać system w oparciu o własne pomysły i konstrukcje.

[1]) W związku z zaprzestaniem produkcji mikrokomputera przez TI, w krótkim okresie można go było nabyć — w ramach wyprzedaży — za 50 dol. — przyp. AJP.

Zasadniczą sprawą przed przystąpieniem do prac projektowych, wydaje się poznanie sposobu organizacji pamięci wewnętrznej w systemie. Struktura ta jest przykładem zastosowania procesora 16-bitowego w środowisku 8-bitowym.



Struktura pamięci wewnętrznej mikrokomputera TI 99/4A

Schemat logiczny organizacji pamięci wewnętrznej przedstawiony został na rysunku 1. Możemy wyróżnić następujące elementy pamięci:
A — blok pamięci ROM o pojemności 2 × 4 KB
B — blok pamięci RAM o pojemności 2 × 128 bajtów
C — główny blok pamięci RAM o pojemności 8 × 16 Kb
D — blok pamięci ROM o dostępie szeregowym.
Poszczególne bloki pamięci pełnią różne funkcje w systemie. Główny blok pamięci ROM (A) zawiera program nadzorujący pracę systemu (MONITOR) oraz część interpretera języka BASIC. Zorganizowany jest on w słowa 16-bitowe i współpracuje bezpośrednio z szynami: danych i adresową (przestrzeń adresowa 4 K słów 16-bitowych); współpracuje on także z pomocniczą pamięcią RAM (blok B), również zorganizowaną w słowa 16-bitowe — dwa układy 6810 produkcji firmy MOTORQLA. Wymienione bloki jako jedyne są bezpośrednio dołączone do szyn: adresowej i danych.

Pozostała część pamięci ma organizację bajtową. Najbardziej oryginalnym elementem jej organizacji jest sposób dołączenia głównego bloku pamięci RAM (C). Składa się on z ośmiu układów pamięci dynamicznych typu 4116 (w sumie 16 KB RAM). Blok C pozostaje pod całkowitą kontrolą układu TMS 9929 (będącego zarazem generatorem obrazu telewizyjnego). Układ ten dostarcza do pamięci RAM dane, sygnały adresowe, a także steruje odświeżaniem pamięci (zgodnie z własnym zegarem, stabilizowanym kwarcem). Procesor komunikuje się z pamięcią wyłącznie za pośrednictwem układu TMS 9929, korzystając z ośmiu bardziej znaczących bitów szyny danych. Systemowa szyna adresowa nie jest w ogóle doprowadzona do tego bloku pamięci. Część RAM-u jest wykorzystywana jako pamięć obrazowa, co zawęża obszar pamięci dostępny dla programów. Procesor ma dostęp do pamięci tylko w momentach, gdy nie jest ona odświeżana lub nie jest wykorzystywana do tworzenia obrazu na monitorze telewizyjnym (przy zapełnieniu ekranu zostawia to zaledwie ok. 10% czasu na dostęp do pamięci przez mikroprocesor). Sposób dostępu procesora do pamięci RAM tłumaczy znaczne spowolnienie wykonywania programów obliczeniowych w stosunku do czasu obliczonego na podstawie cykli instrukcji i okresu zegara.

Pozostała część pamięci (ROM szeregowa oraz złącze użytkownika, umożliwiające rozbudowę pamięci) dołączona jest do systemu przez bufor dwukierunkowy umożliwiający przesyłanie słowa 16-bitowego w postaci dwóch kolejnych bajtów. Rozbudowa pamięci RAM powinna więc być dostosowana do systemu 8-bitowego. Bufor zbudowany jest w oparciu o układy TTL i sterowany również układami TTL. Tak więc czasy przełączania są znacznie krótsze od okresu

zegara systemowego. Do budowy wykorzystano układy 74LS245 bardziej znacząca część szyny danych) oraz 74LS373 i 74LS244 mniej znaczący bajt szyny danych).

Pamięć ROM o dostępie szeregowym (D) współpracuje z układem TMS 9929 (pobierany jest z niego zegar przesuwający) i zawiera przede wszystkim generatory znaków graficznych (standardowych lub specjalnych) oraz procedury opisywane w specjalnym języku GROM (Graphic Read Only Memory Language). Większość modułów programowych dołączanych z zewnątrz również zawiera dodatkowe układy pamięci ROM szeregowej.

Z opisu organizacji pamięci wewnętrznej mikrokomputera TI99/4A wynika, że bez rozbudowy systemu nie jest możliwe zwiększenie szybkości przetwarzania. Ponadto komunikacja z urządzeniami zewnętrznymi w trybie przesyłania równoległego wymaga również dostosowania się do specyfiki systemu. Wydaje się, że istnieją co najmniej trzy sposoby rozbudowy bez angażowania zbyt dużych środków, szczególnie dewizowych:

- rozszerzenie pamięci RAM (sterowanej w klasyczny sposób przez szynę adresową), co jednak pociąga za sobą konieczność dołączenia pamięci szeregowej ROM, zmieniającej opis konfiguracji systemu
- w oparciu o wyprowadzone linie kontrole, danych i adresowe — dołączenie prostego układu z pamięcią EPROM przejmującego kontrolę nad programem nadzorującym
- dołączenie układu pośredniczącego do istniejącego podsystemu przesyłania szeregowego w celu sterowania i współpracy ze standardowymi urządzeniami we-wy dostosowanymi do transmisji szeregowej.

Każdy z wymienionych sposobów zmusza albo do dostarczenia programowi nadzorującemu danych w wymaganej przez niego postaci (konieczność użycia pamięci ROM szeregowej), albo też do przejęcia kontroli nad tym programem (w trybie przerwania lub przez programową zmianę kontekstu). Zamierzeniem niżej podpisanego jest opracowanie urządzenia pośredniczącego, umożliwiającego dołączanie modułów produkowanych przez krajowych wytwórców, jednakże stan zaawansowania pracy nie umożliwia jeszcze pełnej prezentacji.

LESZEK KAMIONKA
Warszawa

Sterowanie napędami dysków elastycznych (2)

Układ WD 1797, proponowany do zastosowania w konstrukcji sterownika napędów dyskowych (mikroKLAN 10), ma pięć rejestrów wewnętrznych wybieranych liniami adresowymi A_1 i A_0 (tab. 1).

Tabela 1

A_1	A_0	ODCZYT RD/=0	ZAPIS WR/=0
0	0	REJESTR STATUSU	REJESTR ROZKAZÓW
0	1	REJESTR ŚCIEŻKI	REJESTR ŚCIEŻKI
1	0	REJESTR SEKTORA	REJESTR DANYCH
1	1	REJESTR DANYCH	REJESTR DANYCH

Wpisując odpowiednie polecenie do rejestru rozkazów — sterujemy wykonywanymi przez układ zadaniami.

RESTORE (pozycjonowanie)

Po otrzymaniu tego rozkazu, FDC 1797 ustawia głowicę zapisująco-odczytującą nad ścieżką 00. Format rozkazu jest następujący (od MSB):

0 0 0 0 h v r_1 r_0

h = 1 gdy przed wykonaniem rozkazu ma nastąpić docisk głowicy R/W do nośnika magnetycznego. Jeśli v = 1, to wykonywany jest odczyt adresu z dysku i dodatkowo sprawdzenie numeru ścieżki w polu adresowym sektora. Bity r_0, r_1 — określają czas pomiędzy poszczególnymi impulsami krokowymi (tab. 2).

Tabela 2

r_1	r_0	Czas przesuwania głowicy	Czas pomiędzy kolejnymi impulsami krokowymi
0	0	3 ms (MAXI)	6 ms (MINI)
0	1	6 ms (MAXI)	12 ms (MINI)
1	0	10 ms (MAXI)	20 ms (MINI)
1	1	15 ms (MAXI)	30 ms (MINI)

MAXI Fclk = 2 MHz
MINI Fclk = 1 MHz

Po zakończeniu wykonywania tego rozkazu układ generuje przerwanie. Odczytując rejestr statusu, możemy sprawdzić poprawność wykonania operacji.

SEEK TRACK (szukaj ścieżki)

Rozkaz ten ustawia głowicę R/W na ścieżce, której numer powinien zostać zapisany w rejestrze danych jako parametr. Układ WD1717 może współpracować z napędami dyskowymi, które mają do 255 ścieżek. Format rozkazu jest analogiczny jak dla RESTORE, a tylko bit czwarty ma wartość "1".

STEP (wykonaj krok)

Po otrzymaniu tego rozkazu układ wysyła jeden impuls (powodujący wykonanie kroku) po linii „step". Wykonanie rozkazu nie zmienia stanu logicznego na linii "direction". Po opóźnieniu określonym przez bity r_1, r_0, jeśli v = 1, wykonywany jest odczyt numeru ścieżki z dyskietki. Jeśli u = 1, to aktualizowana jest wartość Rejestru Ścieżki. Po zakończeniu wykonywania rozkazu generowane jest przerwanie. Format rozkazu jest następujący (od MSB):

0 0 1 u h v r_1 r_0

STEP IN (wykonaj krok do środka)

Po otrzymaniu tego rozkazu układ wysyła impuls kroku, który przesuwa głowicę na następną ścieżkę w kierunku osi obrotu dyskietki. Jeśli u = 1, to zawartość Rejestru Ścieżki jest zwiększana o 1. Pozostałe znaczniki (h, v, r_1, r_0) mają takie same znaczenie jak poprzednio. Format rozkazu:

0 1 0 u h v r_1 r_0

STEP OUT (wykonaj krok na zewnątrz)

Układ wysyła impuls kroku, który przesuwa głowicę na poprzednią ścieżkę (czyli od osi obrotu). Jeśli u = 1, zawartość Rejestru Ścieżki jest zmniejszana o jeden. Format rozkazu:

0 1 1 u h v r_1 r_0

READ SECTOR (czytaj sektor)

Po otrzymaniu tego rozkazu układ wprowadza pole identyfikujące każdego sektora, aż do momentu pobrania prawidłowego numeru ścieżki, sektora, i strony. Wprowadzane jest również CRC. Jeżeli nie zostanie wykryta nieprawidłowość, układ po wprowadzeniu pierwszego bajtu z pola danych ustawia bit w słowie statusu oraz poziom wysoki na linii DRQ, po odczytaniu każdego bajtu z pola danych. Linia DRQ jest ustawiana w stan logiczny 1. Sygnał DRQ ponownie przechodzi w stan 0 w momencie odczytu danej

z Rejestru Danych. Jeśli w odpowiednim czasie nie nastąpi odczyt danej, to w słowie statusu jest ustawiony bit "LATE DMA". Wykonanie rozkazu kończone jest przerwaniem. Format rozkazu:

1 0 0 m F_2 E F_1 0

m = 0 odczyt jednego sektora; m = 1 — odczyt kilku sektorów; F_2 — wskaźnik długości sektora; E = 1 — 15 ms opóźnienia (CLK = 2 MHz); E = 0 — bez opóźnienia; F_1 — wskaźnik wyboru strony: F_1 = 0 — ustawia linię SSO na "0", F_1 = 1 — ustawia linię SSO na "1".

Tabela 3

Typ	Czas pomiędzy zgłoszeniami DRQ (μs)	Czas od DRQ do	
		odczytu (μs)	zapisu (μs)
5,25" FM	64	55	47
5,25" MFM	32	27,5	23,5
8" MFM	16	13	11

Maksymalne czasy przeznaczone na obsługę zgłoszenia DRQ dla zapisu i odczytu różnych dyskietek zostały zestawione w tabeli 3. Przy odczycie kilku sektorów (m = 1), wprowadzanie jest kontynuowane aż do momentu wpisania rozkazu FORCE INTERRUPT do Rejestru Rozkazów.

WRITE SECTOR (zapisz sektor)

Rozkaz ten wykonywany jest podobnie do rozkazu odczytu sektora aż do momentu odnalezienia ID odpowiedniego sektora. Następnie wykonywany jest zapis zawartości sektora na dysk. Format rozkazu:

1 0 1 m F_2 E F_1 a_0

Wskaźniki m, F_2 E i F_1 mają znaczenie identyczne jak poprzednio. a_0 = 0 — ważny sektor danych (znacznik adresowy Pola Danych — FB_H); a_0 = 1 — usunięty sektor danych (znacznik Adresowy Pola Danych = $F8_H$).

READ ADDRESS (odczytaj adres)

Format rozkazu:

1 1 0 0 0 E 0 0

Wskaźnik E ma znaczenie jak poprzednio. Układ odczytuje 6-bajtową zawartość pola ID. W kolejności odczytane są: numer ścieżki, numer strony, numer sektora, długość sektora, CRC 1 bajt, CRC 2 bajt.

READ TRACK (odczytaj ścieżkę)

Format rozkazu:

1 1 1 0 0 E 0 0

Układ odczytuje zawartość całej ścieżki — począwszy od poprzedniego zbocza sygnału "INDEX PULSE". Odczyt kontynuowany jest aż do pojawienia się po raz drugi sygnału "INDEX PULSE".

WRITE TRACK (zapisz ścieżkę)

Format rozkazu:

1 1 1 1 0 E 0 0

Rokaz ten jest wykorzystywany w momencie formatowania dyskietki. CPU musi wpisać do układu (w odpowiedniej formie) zawartość całej ścieżki, bajt po bajcie, w odpowiedzi na żądanie zgłaszane linią DRQ.

FORCE INTERRUPT (wymuś przerwanie)

Format rozkazu:

1 1 0 1 I_3 I_2 I_1 I_0

Bity I_0...I_3 definiują warunki kiedy ma być generowane przerwanie:
I_0 = 1 — w momencie przejścia linii READY z poziomu niskiego na wysoki
I_1 = 1 — z poziomu wysokiego
I_2 = 1 — w momencie pojawienia się impulsu "INDEX"
I_3 = 1 — natychmiast po odebraniu tego rozkazu.
Jeśli I_3...I_0 = 0 to układ jest zerowany bez zgłaszania przerwania.

REJESTR STATUSU

Zawartość tego rejestru pozwala stwierdzić poprawność wykonania rozkazu. Opis poszczególnych bitów Rejestru Statusu został przedstawiony w tabeli 4.

Tabela 4

Bit	Znaczenie
7	Gotowość napędu dyskowego (zanegowany stan linii „READY")
6	Zapis wzbroniony (zanegowany stan linii WRTPRT/)
5	Stan 1 — gdy głowica dociśnięta do nośnika (iloczyn logiczny HLD i HLT)
4	Stan 1 oznacza, że ścieżka, sektor lub numer strony — nie zostały odnalezione
3	Błąd CRC w polu ID
2	TRK00 — przy rozkazach przesunięcia głowicy „LOST DATA" — brak odpowiedzi na sygnał DRQ w odpowiednim czasie dla pozostałych rozkazów
1	DRQ — zgłoszenie żądania odczytu—zapisu INDEX — dla rozkazów przesunięcia głowicy
0	Zajętość — oznacza, że układ jest w trakcie wykonywania rozkazu

OPROGRAMOWANIE

Program sterujący został umieszczony w pamięci od adresu 8000H. Adresy i nazwy symboliczne poszczególnych rejestrów zostały zdefiniowane następująco:

```
COMREG   EQU   0E000H
TRKREG   EQU   0E001H
SECREG   EQU   0E002H
DATREG   EQU   0E003H
SELREG   EQU   0E004H
STANREG  EQU   SELREG
STATREG  EQU   COMREG
```

Aby odczytać lub zapisać sektor, należy ustawić parametry dla tych operacji. Parametry przenoszone są przez rejestry wewnętrzne mikroprocesora. I tak:
Rejestr B — 01 — odczyt sektora; 02 — zapis sektora
Rejestr C — bity b_1 i b_0 określają numer kieszeni napędu: sekwencja 00 — kieszeń 0, sekwencja 01 — kieszeń 1 itd. Gdy b_4 = 1, realizowane jest kodowanie FM, a gdy b_4 = 0 — kodowanie MFM. Bit b_5 = 1 określa współpracę z dyskietkami 5,25", natomiast b_5 = 0 z dyskietkami 8".
W rejestrze D podawany jest numer ścieżki. **W rejestrze E** — numer sektora. W parze **rejestrów HL** podawany jest adres DMA, czyli adres, z którego mają być pobrane dane do zapisu lub gdzie wpisano odczytane dane. Natomiast po zakończeniu realizacji rozkazu, gdy A = 00, to operacja została wykonana poprawnie, natomiast gdy A = FF_H, to operacja została wykonana błędnie.
Podprogram zapisu sektora na dyskietkę może wyglądać następująco (w parze rejestrów HL znajduje się adres obszaru pamięci zawierającego dane przeznaczone do zapisu):

```
WRTFLP: MOV   A,E          ; numer sektora
        STA   SECREG
        LXI   B,DATREG     ; adres portu danych
        LXI   D,STANREG    ; adres zewn. statusu
        MVI   A,0ACH
        STA   ODMREG       ; wpisz rozkaz do FDC
MLLP:   LDAX  D
        RRC                ; czy DRQ = 1?
        JNC   MLLP
        MOV   A,M          ; kolejny bajt danych
        STAX  B
        INX   H            ; zwiększ adres DMA
        JMP   MLLP
```

Zapis realizowany jest w pętli, aż do momentu wygenerowania przerwania kończącego wykonywanie podprogramu zapisu sektora. Podprogram obsługi przerwania wygląda następująco:

```
INTRUT: POP   PSW          ; powrót nastąpi w miejsce skąd wywołano
                           WRTFLP
        LDA   STATREG
```

```
        DB      0EDH,56H  ; kody informujące o za-
                          ; kończeniu obsługi przer-
                          ; wania
        EI
        RET
```

Podobnie można napisać pozostałe procedury obsługi operacji dyskowych. Na wydruku 1 przedstawiono kod oprogramowania umieszczonego w pamięci od adresu 8000H.

```
8000 3E D0 32 00 E0 3A 38 00
8008 32 3C 80 3E C3 32 38 00
8010 E5 2A 39 00 22 3D 80 21
8018 66 81 22 39 00 F3 ED 56
8020 E1 3A 00 E0 FB CD 40 80
8028 F3 ED 46 F5 3A 3C 80 32
8030 38 00 E5 2A 3D 80 22 39
8038 00 E1 F1 C9 00 00 00 0C
8040 78 B7 C8 AF 32 DC 80 E5
8048 D5 16 00 79 E6 03 5F 21
8050 AB 80 19 D1 79 E6 F0 B6
8058 4F 32 6E 81 32 04 E0 E1
8060 C5 01 00 1C 0B 78 B1 C2
8068 64 80 C1 3A 6E 81 4F CD
8070 00 81 E6 80 C2 6B 80 3A
8078 01 E0 FE FF CA 91 80 BA
8080 CA B0 80 E5 D5 C5 CD 0A
8088 81 C1 D1 E1 E6 98 CA B0
8090 80 E5 D5 C5 CD 50 81 C1
8098 D1 E1 3A DC 80 3C 32 DC
80A0 80 FE 0A DA 83 80 3E FF
80A8 B7 37 C9 01 02 04 08 00
80B0 78 FE 02 CA DE 80 3A 6E
80B8 81 4F E5 D5 C5 CD 36 81
80C0 C1 D1 E1 E6 9C C8 E6 10
80C8 C2 91 80 3A DC 80 3C 32
80D0 DC 80 FE 0A DA 83 80 3E
80D8 FF B7 37 C9 00 00 3A 6E
80E0 81 4F E5 D5 C5 CD 1C 81
80E8 C1 D1 E1 E6 FC C8 E6 10
80F0 C2 91 80 3A DC 80 FE 04
80F8 DA 91 80 3E FF B7 37 C9
8100 E5 D5 C5 CD 0A 81 C1 D1
8108 E1 C9 CD 5A 81 7B 32 02
8110 E0 7A 32 03 E0 3E 1F 32
8118 00 E0 FB 76 7B 32 02 E0
8120 01 03 E0 11 04 E0 3E AC
8128 32 00 E0 1A 0F D2 2B 81
8130 7E 02 23 C3 2B 81 7B 32
8138 02 E0 11 03 E0 01 04 E0
8140 3E BC 32 00 E0 0A 0F D2
8148 45 81 1A 77 23 C3 45 81
8150 CD 5A 81 3E 0F 32 00 E0
8158 FB 76 C5 01 F4 01 0B 78
8160 B1 C2 5E 81 C1 C9 F1 3A
8168 00 E0 ED 56 FB C9 00 00
8170 00 00 02 02 00 02 02 07
```

Kod programu sterującego operacjami dyskowymi

W zależności od typu stosowanego napędu i rodzaju zapisu, należy przyjąć:
— częstotliwość zegara 2 MHZ (Z80 CPU) dla 5,25″ i pojedynczej gęstości
— częstotliwość zegara 4 MHz (Z80A CPU) dla 5,25″ i podwójnej gęstości zapisu lub 8″ i pojedynczej gęstości zapisu
— częstotliwość zegara 6 MHz (Z80B CPU) dla 8″ i podwójnej gęstości zapisu.

MIROSŁAW PACZEŚNY
CSK Gdynia

Co? Gdzie?

Podczas 56 Międzynarodowych Targów Poznańskich (p. str. 24) można było znaleźć wiele towarów ułatwiających pracę przy budowie, uruchamianiu i naprawach sprzętu cyfrowego, zwłaszcza mikrokomputerów. Oto ciekawsze z nich, opisane na podstawie materiałów firmowych:

Impulsatory logiczne

Przypominają one znane w krajach zachodnich urządzenia „current pulser" do pobudzania sieci układów logicznych, bez rozcinania połączeń na płytkach drukowanych. Impulsatory logiczne umożliwiają krótkotrwałe (200—700 ns) wymuszenie nowego stanu logicznego, niezależnie od sposobu połączenia sieci logicznej. Skutki tej chwilowej zmiany można zaobserwować za pomocą sondy logicznej, pozwalającej na wydłużenie impulsów.

Logiczny Próbnik Impulsator typu LPI-145 nie tylko umożliwia wymuszenia zmiany stanu logicznego, ale może również zastąpić sondę logiczną. Informacje: Przedsiębiorstwo Techniczno-Handlowe KABIDEZ; 03-468 Warszawa, ul. Stalingradzka 29/31; tel. 11-08-48.
Impulsator Logiczny typu LI-SI uzupełnia sondę logiczną typu SLS-1. Informacje: Zakłady Urządzeń do Montażu Podzespołów Elektronicznych UNITRA-CEMI; 12-100 Szczytno, ul. Dąbrowskiego 6; tel 32-81.

Analizator stanów logicznych

Analizator typu E-220 charakteryzuje się częstotliwością próbkowania do 20 MHz i pozwala na wykrywanie krótkich impulsów (ok. 10 ns). Przyjmuje sygnały o poziomach TTL i ma osiem wejść danych. Zapis jest asynchroniczny lub synchroniczny; można dokonać wyboru zbocza aktywnego. Sygnał zegarowy — zewnętrzny lub wewnętrzny (20 MHz-50 Hz). Zapisywanie danych pod kontrolą sygnałów START i TRIGGER, który można określać jako 8-bitowe słowo. Dane mogą być rejestrowane przed i po wyzwoleniu, a opóźnienie zapisu można nastawiać w zakresie 0—9999 okresów zegara. Pamięć — 256 bajtów. Wyświetlanie na oscyloskopie lub monitorze telewizyjnym. Informacje: Przedsiębiorstwo Doświadczalno-Produkcyjne Elektronicznej Aparatury Pomiarowej EUREKA; 00-277 Warszawa, ul. Freta 39; tel. 31-32-85.

Zestaw odlutowniczy

Jest to narzędzie o nazwie BTT-101 przydatne do wylutowywania podzespołów elektronicznych z płytek drukowanych. Grot został tak zbudowany, że możliwe jest odsysanie lutowia lub wydmuchiwanie go z naprawianego połączenia. Dzięki wbudowanej pompie, nie jest wymagana instalacja sprężonego powietrza. Informacje: Instytut Komputerowych Systemów Automatyki i Pomiarów; 51-608 Wrocław, Al. Młodej Gwardii 1c; tel. 48-10-81 w. 256.

Koszulki izolacyjne termokurczliwe

Produkcji krajowej. Informacje: Zakład Urządzeń Technologicznych; 77-300 Człuchów, ul. Sienkiewicza 21; tel. 771.

Kasety w standardzie Eurokarty

Są one zgodne z normą SEW-3266-81 oraz DIN 41404. Mogą pomieścić 21 modułów o szerokości 20,32 mm i nadają się do montażu w stojakach 19". Istnieją trzy wersje, dla różnych wymiarów płytek drukowanych:
— 0.11.01.34 dla płytek drukowanych o wymiarach 100 × 160 mm
— 0.21.01.34 — o wym. 160 × 233,35 mm
— 0.22.01.34 — o wym. 220 × 233,35 mm
Informacje: Zakłady Urządzeń Przemysłowych POLON; 30-133 Kraków, ul. Dzierżyńskiego 124; tel. 37-39-00.

Zasilacze do systemów cyfrowych

Zasilacze stabilizowane do mikrokomputerów:
EZP 01-00 z wyjściem 5 V/40A
EZP 01-02 — 12V/20A
EZP 01-04 — 24V/10A
EZP 02-10 — 5V/20A
EZP 02-11 — 12V/10A
EZP 02-14 — 24V/5A
EZP 06-00 — 5V/20A
EZP 07-00 — 5V/40A
EZP 08-01 — 5V/10A, +12V/4A, —12V/1A
EZP 08-02 — 5V/10A, +15V/2A, —15V/2A
EZP 04-01 — 5V/20A, +12V/2A, —12V/2A
EZP 04-02 — 5V/20A, +15V/1A, —15V/1A
EZP 05-01 — 5V/20A, +12V/2A, —12V/1A, —5V 1A.

Informacje: Zakłady Automatyki Przemysłowej; 63-400 Ostrów Wielkopolski, ul. Krotoszyńska 35; tel. 624-21.

Zasilacze sieciowe stabilizowane:
SPS 1A-5.20 SC z wyjściem 5V/20A
SPS 1C-5.40 SC — 5V/40A
MPS-150-3/2 — 5V/20A, +12V/2A, —12V/2A
MPS-150-3/3 — 5V/20A, +15V/1A, —15V/1A
MPS-150-3/4 — 5V/20A, +12V/2A, —5V/1A
MPS-120-3/2 — 5V/10A, +12V/4A, —12V/1A
MPS-120-3/3 — 5V/10A, +15V/2A, —15V/2A
MPS-150-4/2 — 5V/20A, +12V/2A, —12V/1A, —5V 1A
MPS-150-4/4 — 5V/20A, +15V/1A, —15V/1A, —5V 1A.

Informacje: Zakład Doświadczalny Elektroniki i Mechaniki Precyzyjnej Politechniki Śląskiej; 44-100 Gliwice, ul. Bałtycka 8; tel. 31-80-81.

Przetworniki DC/DC

Przetworniki napięcia stałego na stałe (z izolacją galwaniczną) są przeznaczone do wytwarzania napięć pomocniczych, przydatnych, na przykład, do zasilania modułów we-wy. Są one zasilane z napięcia głównego +5 V i można je montować bezpośrednio na obwodach drukowanych. Zakład Dowiadczalny Elektroniki i Mechaniki Precyzyjnej opracował takie przetworniki, które z napięcia +5 V dają stabilizowane napięcia +12 V i —12 V lub +15 V i —15 V, o obciążalności od 100 do 200 mA na każdy biegun strony wtórnej.

Niestabilizowaną przetwornicę +5 V/±12 V o obciążalności 2 × 40 mA opracował Ośrodek Badawczo-Rozwojowy Elektronicznych Układów Specjalizowanych MERA-OBREUS; Informacje: Toruń, ul. Grudziądzka 46; tel. 330-45.

Nowoczesne środki chemiczne dla elektroniki

Preparaty klejowe do łączenia elementów wykonanych z różnych materiałów, regeneracji i korekty obwodów drukowanych, ekranowania i odprowadzania ładunków elektrostatycznych. Są to: THERMOCON — kleje elektroizolacyjne przewodzące ciepło, ELEPOX — kleje przewodzące prąd elektryczny oraz ELECTROCON — lakiery przewodzące. Informacje: Przedsiębiorstwo Polonijno-Zagraniczne AMEPOL; 01-126 Warszawa, ul. Brylowska 8a; tel. 32-73-66.

JACEK ŻEBROWSKI

Choć napływają do nas zewsząd prośby o pomoc dla mikroinformatycznych nowicjuszy, nie jesteśmy w stanie zastąpić Wydawnictw Naukowo-Technicznych. Nie zamierzamy spełniać funkcji substytutu książek, które — wierzymy — kiedyś wreszcie zostaną wydane.

Łamy mikroKLANU adresujemy do stawiających drugi i dalsze kroki. Zatem do lektury AKADEMII zapraszamy tych, którzy już przeczytali instrukcję obsługi posiadanego mikrokomputera (praktycznie zawsze można tam znaleźć lepszy lub gorszy opis zastosowanej wersji języka BASIC).

Eleganckie programowanie, które chcemy lansować w AKADEMII mikroKLANU pozwala na rzecz chyba najważniejszą — wielokrotne wykorzystanie. Jak wykazuje praktyka — program, który okazał się użyteczny, prawie zawsze będzie wymagał w przyszłości zmian. Aby można je było wprowadzić, trzeba dokładnie wiedzieć, co robią poszczególne bloki programu.

AKADEMIA mikroKLANU

Czy potrafimy programować elegancko? Elegancja wyszła jakoś z mody i w rezultacie mamy szeroką rzekę programów, które często są nierozumiałe nawet dla autorów. Ludzie wymyślili języki programowania wysokiego poziomu po to, aby programy napisane wczoraj mogły być zrozumiane jutro. Również przez innych.

Wprawdzie E. W. Dijkstra (którego książkę — „Umiejętność programowania", WNT, 1978 — wszystkim polecamy) ostro potępia BASIC jako najgorszy język programowania mikrokomputerów i nic na to nie poradzimy. Nasz kurs eleganckigeo programowania będzie więc oparty na standardowym BASICU (p. INFORMATYKA 5/83).

Dziś pierwszy odcinek, a w nim — test komputera. Niestety, nasza Akademia nie może zapewnić kursantom pomocy naukowych...

WŁAŚCIWE WYJŚCIE

Każdy kurs programowania zaczyna się od pytania: „Jaki element programu jest niezbędny?". Wszyscy kursanci zapewne krzykną tu chórem: „Wyjście!!". I będą mieli rację, choć jest to całkiem odwrotnie niż w życiu codziennym (gdy jest alternatywa, ale nie ma wyjścia). W programowaniu nie ma alternatywy — musi być wyjście, gdyż informuje nas ono o tym, **co** dany program robi (lub zrobił). Najgorszym możliwym wyjściem jest wyjście typu:

```
RUN
M= 1.23
X=—1
READY.
```

Może być ono wystarczające dla autora programu (który wie, że **M** jest liczbą nosorożców, a **X** — temperaturą), ale innemu użytkownikowi nic to nie mówi. Tym bardziej, gdy — jak w naszym przykładzie — liczba nosorożców wychodzi niecałkowita, a temperatura ujemna. Takie wyjście nie tylko nie pozwala sprawdzić, czy program działa poprawnie, ale co gorzej — przekazuje informację niepełną, czyli dezinformuje. Co należy robić, aby wyjście było czytelne? Pisać, pisać, pisać! Najlepiej pełnymi zdaniami, bez skrótów i uproszczeń. Zaprocentuje nam to już za pół roku, gdy bez kłopotów będziemy rozumieli własny program.

Pierwszy „szkolny" program przedstawiono na wydruku.

```
LIST

5 REM PROGRAM TESTUJACY DOKLADNOSC
6 REM OBLICZEN WEWNETRZNYCH KOMPUTERA.
10 N=-2
20 X=1
30 Y=81
40 Y=X/Y
50 Y=Y*10
60 X=INT(Y)
70 READ A
80 N=N+1
90 Y=Y-A
100 IF X=A THEN GOTO 50
110 PRINT "TEN KOMPUTER WYKONUJE OBLICZENIA NA "!N!" MIEJSCACH DZIESIETNYCH"
120 DATA 0,1,2,3,4,5,6,7,9,0,1,2,3,4,5,6,7,9,0,1
130 END
READY.
     RUN
NEPTUNE 104 WYKONUJE OBLICZENIA NA  10  MIEJSCACH DZIESIETNYCH

READY.
```

Zwróćmy uwagę, że każdy program powinien zawierać na początku choć jedną linijkę komentarza (u nas linie 5 i 6), objaśniającego **co** program robi — samo wyjście może być niezbyt zrozumiałe!

Parę uwag praktycznych. Dla większej przejrzystości opuszczamy w naszych programach słówko LET — w rzeczywistości wiele interpreterów BASIC wymaga podania tego słowa w tekście programu (np. ZX SPECTRUM). Każdy program powinien mieć też zakończenie. W niektórych wersjach języka BASIC (jak we wspomnianym już SPECTRUM) nie ma instrukcji END — wtedy należy użyć instrukcji STOP. Przestrzeganie tego polecenia stanowi przynajmniej częściowe zabezpieczenie przed programami zapętlonymi. Wprawdzie dla mikrokomputerów takie zapętlenie nie jest zbyt groźne (zawsze można wyłączyć komputer), ale elegancja wymaga, by użytkownik wiedział gdzie kończy się program.

Wreszcie uwaga końcowa — wszyscy dysponujący komputerami o rozszerzonym BASICU, który pozwala używać zmiennych wieloliterowych, powinni nadawać zmiennym sensowne i zrozumiałe nazwy. Coś podobnego zostało zrobione w naszym programie — N zazwyczaj oznacza liczbę zdarzeń; X, Y, Z używamy do opisu zmiennych (to pamiątka z przeszłości, gdy liczyliśmy na kalkulatorze), a stałe oznaczamy A, B, C itd.

Już słychać głośne BEEP; cóż, lekcja skończona. Do następnego spotkania!

JAKUB TATARKIEWICZ

---

prowadzi
ANDRZEJ J. PIOTROWSKI
tel. dom. 48-22-85

Dokończenie ze str. 12

Ze względu na czytelność programu oraz potrzebę ustalania nazw zgodnych z terminologią zakresu zastosowań CHILLA, bardzo użyteczny jest również typ wyliczeniowy (ang. set mode). Przy użyciu typu wyliczeniowego ustala się zestaw nazw oznaczających wartości, które może przyjmować zmienna tego typu, podobnie do literałów dla typów standardowych. Zastosowanie typu wyliczeniowego zilustrowano poniższym przykładem.

```
1. NEWMODE LINE_STATES SET(ON_HOOK, OFF_HOOK, BU-
   SY, IDLE, DISABLED);
2. DCL CURRENT_STATE LINE_STATES := ON_HOOK;
   ...
3. IF CURRENT_STATE = BUSY
4.   THEN GIVE_BUSY_TONE();
5. FI;
```

Do opisu stanów linii telefonicznej użyto zadeklarowanej w linii 2 zmiennej CURRENT_STATE typu LINE_STATES, zdefiniowanego w linii 1. Definicja określa pięć różnych wartości, które może przybierać zmienna inicjowana wartością ON_HOOK.

Jeśli nie jest konieczne operowanie pełnym zestawem wartości typu wyliczeniowego, to zaleca się stosowanie typu okrojonego (ang. range mode) — ze względu na możliwość intensywniejszej kontroli typów. Typ okrojony powstaje przez zdefiniowanie nowego typu z okrojonym zbiorem wartości względem typu macierzystego, np.:

```
NEWMODE BYTE = INT(0 : 255),
            NORMAL_STATES = LINE_STATES(ON_HOOK:IDLE);
```

Zestaw stanów linii telefonicznej opisany uprzednio typem LINE_STATES został w definicji typu okrojonego NORMAL_STATES ograniczony do czterech wartości od ON_HOOK do IDLE. Stosowanie typu okrojonego pozwala nie tylko na zwiększenie niezawodności, lecz także na uzyskanie programu efektywniejszego pod względem zajętości pamięci czy szybkości wykonania.

Typy mające postać ciągów (ang. string mode), tzn. typ znakowy (ang. character string) i typ bitowy (ang. bit string) uzupełniono możliwościami dokonywania operacji na ciągach. Dostęp do części ciągu znakowego lub bitowego umożliwiają operacje indeksowania (ang. indexing) i wyodrębiania (ang. substringing), np.:

```
DCL OVERFLOW_INDICATOR BIT(1) := STATUS_WORD(0),
    CONDITION_CODE BIT(3) := STATUS_WORD(2:4);
```

Zmienna OCERFLOW_INDICATOR zadeklarowana jako ciąg jednobitowy jest inicjowana wartością bitu wskazanego indeksem 0 w ciągu bitowym STATUS_WORD. Bity o indeksach 2 do 4 tego ciągu są przypisane zmiennej CONDITION_CODE, zadeklarowanej jako ciąg trzybitowy.

Operacjami dokonywanymi na ciągach są także porównywanie (ang. comparison) i seklejanie (ang. concatention). Ponadto ciągi bitowe mogą być poddawane operacjom logicznym wykonywanym na parach bitów wyznaczanych przez indeksowanie.

W CHILLU przewidziano różnorodne możliwości używania zmiennych typu wskaźnikowego (ang. reference mode). Wartościami wskaźników są adresy lub deskryptory wskazywanych obiektów. Przewidziano wskaźnik pusty, któremu przypisywany jest specjalny literał wskaźnikowy **NULL**. Przypisanie wartości wskaźnikiem dokonuje się przy użyciu jednoargumentowego operatora "—>".

W celu umożliwienia efektywniejszej kontroli typów rozróżniono typ wskaźnikowy związany (ang. bounded) i swobodny (ang. free). Wskaźnik związany wskazuje tylko zmienne jednego, zadeklarowanego typu, natomiast swobodny nie jest związany z typem wskazywanej zmiennej. Wskaźnik związany jako bardziej ograniczony zmniejsza niebezpieczeństwo błędu, natomiast wskaźnik swobodny pozwala uzyskać większą elastyczność.

Zastosowanie wskaźników zilustrowano w poniższym przykładzie obsługi struktur listowych.

```
1. CIRCULAR_LIST:
2. MODULE
3.   (* DWUSTRONNA LISTA CYKLICZNA *)
4.   NEWMODE NODE = STRUCT(PRED, SUC REF NODE,
                                    VALUE INT);
     LUE INT);
5.   DCL HEAD NODE := (: NULL, NULL, 0 :);
6.   DCL POOL ARRAY(1 : 1000) NODE;
7.   DCL LAST REF NODE := —> HEAD;
8.   DO FOR NEW IN POOL;
9.     NEW.PRED := LAST;
10.    LAST —>.SUC := — NEW;
11.    LAST := —> NEW;
12.    NEW.VALUE := 0;
13.  OD;
14.  HEAD.PRED := LAST;
15.  LAST —>.SUC := —> HEAD;
16. END CIRCULAR_LIST;
```

W linii 4 zdefiniowano typ NODE elementów listy jako strukturę o trzech polach. Dwa z nich są wskaźnikami związanymi z typem NODE, wskazującymi poprzednik elementu listy PRED i jego następnik SUC. Trzecie pole jest przeznaczone na daną VALUE typu **INT**. Pierwszy element listy HEAD zadeklarowano w linii 5, a w linii 6 — tablicę zawierającą 1000 elementów typu NODE. W pętli zawartej w liniach 8—13 tworzona jest lista przez połączenie wszystkich elementów NODE przy użyciu zmiennej pomocniczej LAST oraz licznika pętli NEW. Zamykające listę połączenie ostatniego elementu NODE z jej nagłówkiem następuje w liniach 14 i 15.

W powyższym przykładzie użyto wskaźników związanych, ponieważ wszystkie elementy listy są tego samego typu. Gdyby istniała taka potrzeba, zastosowanie wskaźników swobodnych umożliwiłoby obsługę listy składającej się z obiektów dowolnego typu.

Omawiane dotychczas rodzaje wskaźników odnosiły się do typów statycznych. Natomiast do zmiennych typu dynamicznego — np. dynamicznych ciągów, tablic lub struktur — stosuje się wskaźniki typu szeregowego (ang. row mode). Wskaźniki te są szczególnie efektywne przy operowaniu tekstami oraz przy wszelkich operacjach na ciągach i ich fragmentach. Rozpatrzymy na przykład — zastosowanie typów szeregowych do manipulowania ciągami znakowymi:

```
DCL M CHAR(5) := 'HITCH',
    ROW CHAR(5) := —>M;
R —> (1) := 'D';
```

Zmienna M zadeklarowana jako ciąg pięcioznakowy jest inicjowana napisem 'HITCH'. Zmienna R jest zadeklarowana jako wskaźnik szeregowy, opisujący dowolny ciąg, złożony co najwyżej z pięciu znaków. Po nadaniu wartości początkowej, zmienna R opisuje wszystkie elementy M. Sposób dostępu do tych elementów przedstawiono przez nadanie pierwszemu z nich wartości D. Zmiennej R można nadać inne znaczenie związując wskaźnik szeregowy z elementami 2, 3 i 4 ciągu M:

```
R := —>M(2 : 4);
```

Konstrukcja M (2 : 4) nazywana jest podciągiem (ang. substring) lub wycinkiem (ang. string slice).

Typ danych może być zaopatrzony w atrybuty. Jednym z atrybutów w CHILLU jest niezapisywalność zmiennych (ang. read only location), oznaczającą ich dostępność tylko do odczytu. Zmienna opatrzona artybutem **READ** uzyskuje wartość wskutek jej inicjowania, a potem nie może być modyfikowana. Zmienna złożona może być w całości niezapisywalna lub poszczególne jej składniki mogą być zabezpieczone przed zapisem, np.:

```
DCL X READ ARRAY(1 : 3) INT := (:5,9,7:);
    Y READ STRUCT (B BOOL, C CHAR) := (:TRUE, 'A':),
    Z STRUCT (D BOOL, E READ CHAR) := (:FALSE, 'B':);
```

Wszystkie elementy tablicy X i pola struktury Y są dostępne wyłącznie do odczytu. Natomiast struktura Z posiada tylko jedno pole niezapisywalne. Atrybut niezapisywalności zastosowano jako środek zwiększania niezawodności programów, dlatego też stosowanie go zalecane jest we wszystkich możliwych przypadkach.

Przy opisie struktur danych występuje często potrzeba użycia obiektów podobnych do siebie, ale różniących się na przykład szczególnym polem w strukturze. Takie obiekty nazwano strukturami wariantowymi (ang. cariant structures). Jako przykład weźmy strukturę listową buforów o różnej długości zdefiniowanych jako struktury wariantowe:

```
1. SYN SMALL_SIZE = 4,
2.     LARGE_SIZE = 12;
3. NEWMODE BUFFER_SIZES = SET(SMALL,LARGE);
```

```
4. NEWMODE BUFFERS = STRUCT(NEXT REF BUFFERS)
5.                          SIZE BUFFER_SIZES,
6                           CASE SIZE OF
7.                           (SMALL): S_AREA CHAR
                                      (SMALL_SIZE),
8.                           (LARGE): L_AREA CHAR
                                      (LARGE_SIZE),
9.                          ESAC
10.                         );
```

W linii 1 zdefiniowano synonimy SMALL_SIZE i LARGE_SIZE, a w linii 3 typ wyliczeniowy BUFFER_SIZES ustalający nazwy literałów SMALL i LARGE. W liniach 4 do 10 zawarta jest definicja buforów jako nowego typu BUFFERS będącego strukturą wariantową. Pierwsze pole tej struktury o nazwie NEXT wskazuje następny bufor listu. Pole to jest typu **REF** BUFFERS, może więc wskazywać zarówno długi, jak i krótki bufor. Drugie pole — SIZE typu BUFFER_SIZES — jest znacznikiem długości bufora. Takie pole w strukturze wariantowej nosi nazwę pola znacznikowego (ang. tag field). Konstrukcja **CASE_ESAC** w liniach 6—9, będąca członem definicji, a nie instrukcją, jest nazywana częścią wariantową (ang. variant part) struktury. Poszczególne warianty opisane są przez alternatywy członu **CASE**, przy czym selektorem alternatywy jest wartość pola znacznikowego. W podanym przykładzie altertywami są różne obszary bufora, które są ciągami znakowymi o długości 4 lub 12 znaków, nazwane odpowiednio S_AREA i L_AREA. Nazwy te są podstawiane pod nazwy pola struktury.

W CHILLU przewidziano możliwości niskiego poziomu takie jak zmiany gęstości upakowania i kontrola ułożenia danych. Słowa kluczowe **PACK** i **NOPACK** w opisie typu specyfikują, czy wymagane jest gęste czy mniej gęste niż standardowe upakowanie danych. Niekiedy niezbędna jest bezpośrednia kontrola ułożenia danych, w związku z tym w CHILLU przewidziano możliwość dokonywania opisu ich rozmieszczenia (ang. layout description). Polega on na jawnym wskazaniu pól i elementów danych (np. struktur czy tablic) wewnątrz słowa maszynowego. Tak precyzyjny opis jest niezbędny przy sprzęganiu struktur danych utworzonych poza programem w języku asemblera. Korzystanie z tych możliwości języka zaleca się jedynie w przypadkach rzeczywiście niezbędnych, albowiem opis danych jest silnie maszynowo uzależniony.

PRZYKŁAD PROGRAMU W JĘZYKU CHILL

Na zakończenie trzech odcinków opisu CHILLA przedstawimy przykład zastosowania tego języka do programowania systemów komutacyjnych. Fragment programu, prezentujący charakterystyczne właściwości CHILLA, podał Kristen Rekdal [3].

```
1. MODULE
2.     SEIZE ON_HOOK, OFF_HOOK, ABORT;
3.     GRANT DIGIT, DIALLED_NUMBER;
4.     SIGNAL DIGIT = (CHAR ('0':'9')),
           DIALLED_NUMBEY = (CHAR(N));
6.     SYN N=8;        (*NO. OF DIGITS IN NUMBER*)
7.     DIGIT_RECEPTION;
8.     MODULE
9.     SEIZE OFF_HOOK, NUMER_RECEIVER;
10.     DCL USER INSTANCE;
11.      START_DIGIT_RECEPTION;
12.      DO FOR EVER;
13.      RECEIVE CASE SET USER;
14.          (OFF HOOK): START NUMBER_RECEIVER (USER);
15.      ESAC;
16.      OD START_DIGIT_RECEPTION;
17.     END DIGIT_RECEPTION;
18.     RECEIVE_NUMBER:
19.     MODULE

20.       SEIZE DIGIT, ON_HOOK, ABORT, DIALLED_NUMBER, N;
21.       GRANT NUMBER_RECEVER;
22. NUMBER_RECEIVER:
23. PROCESS (USER INSTANCE);
24.          DCL NUMBER CHAR(N):=(NEI'';
25.          AWAIT_DIGITS:
26.          DO FOR I IN INT (0:N—1);
27.          RECEIVE CASE
28.            (DIGIT IN D): NUMBER (I) :=D;
29.            (ON_HOOKi: SEND ABORT TO USEC;
30.              STOP;
31.          ESAĆ;
32.           OD AWAIT_DIGITS;
33.          SEND DIALLED_NUMBEY (NUMBER) TO USER;
34.       END NUMBER_RECEIVER;
35.   END RECEIVE_NUMBER
36.  END;
```

Przykładowy program w języku CHILL opisuje odbiór ośmiu cyfr od abonenta. Program jest zamknięty w module, który komunikuje się z otoczeniem przy użyciu zestawu sygnałów. Sprzężenie z otoczeniem jest opisane w liniach 2 i 3. W linii 2 dzierżawione są nazwy sygnałów: ON_HOOK, OFF_HOOK i ABORT (przyjmuje się, że te sygnały zdefiniowano poza modułem). Natomiast w linii 3 udostępniono na zewnątrz nazwy sygnałów: DIGIT i DIALLED_NUMBER zdefiniowane w liniach 4 i 5. Sygnał DIGIT zawiera komunikat w postaci parametru znakowego z zakresu cyfr 0—9, a DIALLED_NUMBER — w postaci ciągu N-znakowego. W linii 6 zdefiniowano N jako synonim stałej dla oznaczenia ilości cyfr w numerze i nadano jej wartość N = 8.

Na początku modułu DIGIT_RECEPTION dopuszczono do użycia wewnątrz niego nazwy: OFF_HOOK i NUMBER_RECEIVER. W linii 10 zadeklarowano zmienną USER typu **INSTANCE**, która służy do identyfikowania nadawcy numeru. Odbiór numeru rozpoczyna się w pętli nieskończonej, zawartej w liniach 12—16. Po każdym odebraniu sygnału OFF-HOOK wyzwalany jest proces NUMBER_RECEIVER i następuje powrót do etykiety START_DIGIT_RECEPTION oznaczającej stan gotowości do przyjęcia następnego sygnału. Wówczas wykonanie programu jest wstrzymywane na konstrukcji **RECEIVE CASE** zapisanej w liniach 13—15, aż do nadejścia sygnału OFF_HOOK. Zapis **SET** USER przy instrukcji **RECEIVE CASE** powoduje ustalenie identyfikatora nadawcy sygnału i przypisanie go zmiennej USER.

Moduł RECEIVE_NUMBER zapisano w liniach 18 do 35, a jego sprzężenie z otoczeniem — w liniach 20 i 21. Udostępniona na zewnątrz nazwa NUMBER_RECEIVER oznacza proces zdefiniowany w liniach 22 do 34. Zadaniem tego procesu jest odbiór i gromadzenie N cyfr nadawanego numeru. Parametrem procesu jest wyżej opisana zmienna USER. W linii 24 zadeklarowany jest ciąg N-znakowy NUMBER inicjowany spacjami. Blok AWAIT_DIGITS, rozpoczynający się w linii 25, zawiera pętlę zliczającą N cyfr. Zawarta w niej instrukcja **RECEIVE CASE** odbiera dwa sygnały DIGIT lub ON_HOOK. Gdy nadejdzie sygnał DIGIT, to związana z nim wartość parametru znakowego odbierana jest przez zmienną D, a następnie podstawiana do ciągu NUMBER. Gdy odebranym sygnałem jest ON_HOOK, to wysyłany jest sygnał ABORT do odbiorcy wskazanego wartością identyfikatora USER i proces kończy się w linii 30. Po odebraniu N cyfr, proces zakończy swoje działanie wysłaniem sygnału DIALLED_NUMBER wraz z komunikatem NUMBER zawierającym odebrany numer.

LITERATURA

[1] CCITT: CHILL Langauge Definition. CCITT Recomendation Z200, November 1980

[2] CCITT: Introduction to CHILL. CCITT Manual, May 1980

[3] Rekdal K.: CHILL — The Standard Language for Programming SPC Systems. IEEE Transactions on Communications, June 1982.

Badania międzynarodowe MERY 60

W październiku br. przeprowadzone zostały w Bielsku-Białej badania opracowanego w CNPSS MERASTER oprogramowania mikrokomputera MERA 60 (SM 1633). Do badań przedstawiono dwa pakiety oprogramowania emulacyjnego, które pozwala na zastosowanie mikrokomputera MERA 60 jako terminala inteligentnego dla maszyn Jednolitego Systemu. Oprogramowanie to emuluje funkcje terminala interakcyjnego typu IBM 3270, a także terminala dla pracy wsadowej typu IBM 3780.

Badania przeprowadzono w warunkach rzeczywistej eksploatacji oprogramowania w Ośrodku Wdrażania Postępu Technicznego w Energetyce ZEOPd, gdzie jest ono wykorzystywane od ponad roku. W badaniach brali udział przedstawiciele ZSRR, CSSR, WRL, LRB i PRL.

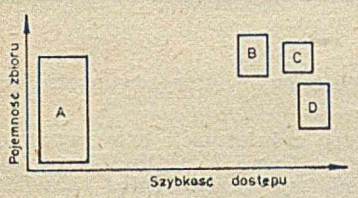
Podpisane akty przeprowadzonych badań rekomendują włączenie badanego oprogramowania do wspólnego banku modułów programowych RWPG.

II/A. Organizacja i funkcjonowanie systemów komputerowych

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

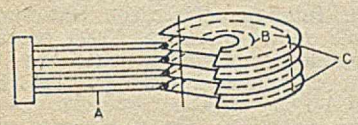
PYTANIA

1. Podaj właściwe identyfikatory (litery) przedstawionych na wykresie obszarów stosowalności — przy wymienionych rodzajach fizycznych nośników danych:



taśma magnetyczna ...
dyski magnetyczne ...
karty dziurkowane ...
pamięć operacyjna ...

2. Dopasuj oznaczenia literowe z rysunku do następujących pojęć:



ścieżka ...
cylinder ...
mechanizm dostępu ...

3. Przyporządkuj najwłaściwsze definicje kolejno wymienionym pojęciom:

Półdupleks to ...
Pełny dupleks to ...
Synchroniczność to ...
Asynchroniczność to ...
Simpleks to ...

A) łącze umożliwiające jednoczesną transmisję w obu kierunkach
B) praca ze zmiennym odstępem czasu między kolejnymi zdarzeniami transmisji
C) praca ze stałym odstępem czasu między kolejnymi zdarzeniami transmisji
D) łącze umożliwiające transmisję w dowolnym kierunku, ale nigdy w obu jednocześnie
E) łącze umożliwiające transmisję wyłącznie w jednym ustalonym kierunku

4. Główną zaletą architektury stosowej (ang. stack-oriented) w systemach komputerowych jest ...

a) możliwość nakładania czasowego (ang. overlap) operacji wewnętrznych (CPU) oraz zewnętrznych (we-wy)
b) brak potrzeby opracowywania asemblera
c) redukcja liczby jawnych odwołań pamięciowych
d) możliwość mikroprogramowania

5. Podstawowa różnica między przerwaniem wewnętrznym (ang. trap) a przerwaniem zewnętrznym (ang. interrupt) sprowadza się do tego, że pierwsze z nich jest ..., podczas gdy drugie jest ...

a) .... kompatybilne (ang. compatible) ... niekompatybilne (ang. uncompatible)
b) .... asynchroniczne ... synchroniczne
c) .... odtwarzalne (ang. reproducible) .... nieodtwarzalne (ang. irreproducible)
d) .... nakładkowe (ang. overlapped) .... nienadakładkowe (ang. unoverlapped)

6. Załóżmy, że instalacja komputerowa obejmuje zarówno szybkie pamięci zewnętrzne (np. dyski), jak i wolnobieżne urządzenia zewnętrzne (np. czytniki kart, drukarki). Czym — mówiąc najogólniej — powinna charakteryzować się strategia kosztowo-wydajnościowej optymalizacji przydziału kanałów?

a) zapewnieniem pierwszeństwa obsługi wszystkim wolnobieżnym urządzeniom, przez przydzielanie im kanałów o najmniejszych numerach
b) rozdzieleniem urządzeń po jednym na każdy kanał, w celu zmniejszenia groźby interferencji zgłoszeń
c) przydzielaniem wszystkich wolnobieżnych urządzeń do (jednego lub więcej) kanałów multiplekserowych
d) przydzielaniem po jednym urządzeniu na każdy kontroler

7. Które z poniższych sformułowań stosunkowo najtrafniej oddaje istotę zjawiska określanego jako „interferacja pamięci" lub „podkradanie cykli"?
a) pobieranie, dekodowanie i „podkradanie" liczb binarnych przechowywanych w pamięci stałej — jest wykonywane przez procesor
b) korzystanie przez sterownik urządzenia zewnętrznego (ang. device controller) z pamięci operacyjnej wymusza na procesorze stan oczekiwania (ang. wait)
c) schemat adresowania pośredniego narzuca na procesor udział w realizacji pobierania kolejnych wartości z łańcucha adresów
d) sterownik urządzenia zewnętrznego wywłaszcza procesor dla zasygnalizowania zakończenia zadania

8. Wymień podstawowe kolejne kroki związane z obsługą przerwania odebranego przez procesor.

9. Rozważ sytuację, w której programista ma napisać program sortujący zbiór w porządku alfabetycznym. Załóżmy, że dane są typu alfanumerycznego, a rekordy mają dziesięcioznakowe klucze alfabetyczne. Na co programista powinien w zasadzie zwrócić największą uwagę?

a) na wewnętrzną reprezentację liczb stałoprzecinkowych w zbiorze
b) na układ bitów kodujących poszczególne znaki występujące w zbiorze
c) na kolejność dobierania rekordów w maszynie, która będzie realizować sortowanie
d) na procedury konwersji danych alfabetycznych do postaci numerycznej

10. Załóżmy, że w systemie operacyjnym użyto wirtualnego schematu zarządzania pamięcią ze stronicowaniem na żądanie (ang. demand-paged memory management). Przyjmijmy, że w pewnym momencie „mapa stronic" ma następującą postać:

Numer strony wirtualnej	0	1	2	3	4	5	6
Numer bloku pamięci operacyjnej	2	4	/	3	/	0	/
Stan strony	T	T	N	T	N	T	N

Do jakich rzeczywistych słów pamięci nastąpią faktyczne odwołania lub jakie pojawią się inne skutki w wyniku próby wykonania każdej z podanych niżej instrukcji?
a) umieść w akumulatorze zawartość słowa ze strony 0 z przesunięciem 7: ....
b) dodaj do akumulatora zawartość słowa ze strony 2 z przesunięciem 26: ....
c) zapamiętaj zawartość akumulatora w słowie na stronie 1 z przesunięciem 15: ....

ROZWIĄZANIA

1. B, C, A, D; 2. B, C, A; 3. D, A, C, B, E; 4. c; 5. c (Uwaga: Niektórzy producenci sprzętu uważają pojęcia „interrupt" oraz „trap" za synonimy); 6. c; 7. b; 8. Zapamiętanie stanu przerwanego programu, przejście do podprogramu obsługi przerwania, obsługa przerwania, odtworzenie stanu przerwanego programu,

LITERATURA

ad 1. Gear W.: Computer Organization and Programming, II wyd., McGraw-Hill, p. 238
ad 2. Sanders D.: Computers in Business. McGraw-Hill, p. 194—197
Bohl M.: Information Processing. Science Research Associates, p. 80—3
ad 3. Watson R.: Timesharing System Design Concepts. McGraw-Hill, p. 100—103
Martin J.: Telecommunications and the Computer. Prentice-Hall, p. 438—457
ad 4. Wegner P.: Języki programowania, struktury informacji i organizacja maszyny cyfrowej. PWN, 1979, s. 90
ad 5. Tanenbaum A.: Organizacja maszyn cyfrowych w ujęciu strukturalnym. WNT, 1980, s. 170
ad 6. Gear, ibid. p. 254—255
ad 7. Watson, ibid, p. 80—81
Gear, ibid. p. 67
ad 8. Blaauw G.: Digital System Implementation. Prentice-Hall, pp. 220—222
Gear, ibid. p. 87—91
Watson, ibid. p. 218—221
ad 9. Knuth D.: The Art of Computer Programming Vol. 3; Sorting and Searching. Addison-Wesley, ex. 13 p. 7; answer 13 p. 574
ad 10. Madnick S., Donovan J.: Systemy Operacyjne, PWN, 1983.

ZE ŚWIATA

Porozumienie DIGITAL RESEARCH

W maju 1982 r. firma DIGITAL RESEARCH INC. (DRI) była w dużych tarapatach. Chociaż jej 8-bitowy system operacyjny CP/M pozostawał wciąż przemysłowym standardem, rynek przestawił się już na mikrokomputery 16-bitowe, a firma MICROSOFT odniosła znaczny sukces wprowadzając dla komputerów osobistych IBM swój system operacyjny PC-DOS.

W tej sytuacji firma DRI musiała przestawić swą działalność. Zaczyna poprawiać i rozszerzać asortyment — przerabiając większość swych systemów na systemy przenośne. DRI w okresie tym zawarła również kilka porozumień z dużymi firmami produkcji sprzętu komputerowego, telekomunikacyjnego i półprzewodników, które udostępniły jej dużą część rynku. Nie mogąc dogonić firmy MICROSOFT w dostawie oprogramowania dla mikrokomputerów 16-bitowych, DRI zaczęła przygotowywać się do stworzenia oprogramowania użytkowego opartego na systemie operacyjnym UNIX dla mikrokomputerów 32-bitowych.

Dzięki dużemu popytowi na oprogramowanie DRI, mimo trudnej dla siebie sytuacji, dwukrotnie zwiększyła zatrudnienie (do ponad 500 osób), oraz obroty (z 18,5 mln dol. w 1982 do 44,6 mln dol. w 1983 roku). W porównaniu do 70 mln dol. firmy MICROSOFT było to dużo mniej, ale wystarczająco, by pozostać liczącym się na rynku konkurentem.

W 1982 roku kłopoty DRI wynikały w znacznym stopniu z tego że MICROSOFT miała gotowy 16-bitowy system operacyjny PC-DOS dla komputerów osobistych, w czasie gdy projekt jej własnego 16-bitowego systemu CP/M-86 był jeszcze w proszku. Nawet wersja wielozadaniowa (ang. multitasking version) systemu CONCURRENT CP/M, znalazła niewielu odbiorców jako zbyt kosztowna dla komputerów osobistych.

Pozbawiona podstawowego 16-bitowego systemu operacyjnego, DRI zagrożona była utratą poparcia trzeciej części programistów, którzy stworzyli olbrzymi obszar zastosowań dla systemu CP/M. Firma mogła odzyskać utraconą pozycję tylko poprzez stworzenie standardu przemysłowego, odpowiadającego skali popularności komputerów osobistych. Oznaczało to zawarcie porozumienia z IBM i zajęcie się systemem PC-DOS.

To drugie zadanie było stosunkowo łatwe, ponieważ system PC-DOS wywodził się właśnie z systemu CP/M (prezes firmy MICROSOFT, Bill Gates, napisał PC-DOS według zmodyfikowanej wersji CP/M).

W czerwcu 1983 DRI zapowiedziało serię kompilatorów pozwalających działać programom CP/M pod nadzorem PC-DOS i vice versa. Miesiąc później IBM wprowadził na rynek CONCURRENT CP/M jako standardowy system operacyjny dla komputerów osobistych — i to w tej samej cenie co PC-DOS.

W ten sposób DRI znalazła się wśród „16-bitowców". Chcąc zrobić dalszy krok, wykonała własne oprogramowanie graficzne CSX dostępne w systemach operacyjnych PC-DOS i MS-DOS, przy czym w PC-DOS z różnymi strukturami wejścia-wyjścia stosowanymi w niektórych mikrokomputerach podobnych do IBM.

Następnym krokiem było zawarcie porozumienia z firmą INTEL na opracowanie systemu V UNIX dla jej 16-bitowego mikroprocesora oraz iAPX 286.

W styczniu br. DRI podała dwie sensacyjne informacje. Pierwszą z nich była wiadomość o porozumieniu z AT&TTECHNOLOGIES INC. na opracowanie biblioteki programów użytkowych dostosowanych do systemu UNIX i przeznaczonych do zastosowań gospodarczych. Porozumienie to dawało DRI coś co było jej bardzo potrzebne — współpracę z gigantem UNIX, natomiast AT&T — dostęp do ponad 2000 sklepów sprzedaży detalicznej komputerów, autoryzowanych przez DRI.

Drugą z informacji było zawarcie umowy z firmą MOTOROLA na opracowanie oprogramowania pozwalającego na zamienność stosowania systemów operacyjnych UNIX i systemów DRI dla mikrokomputerów opartych na 16/32-bitowym mikroprocesorze 68000. Intencją obu firm jest wyposażenie mikroprocesora 68000 w takie narzędzia programowe, jakie są dostępne dla mikroprocesora INTEL 8080 w IBM PC.

Szczególną wymową w tej zapowiedzi ma fakt odwoływania się nie do systemów UNIX i CP/M, ale do UNIX i CONCURRENT DOS — nazwy nie podawanej przedtem, przynajmniej oficjalnie.

DRI ujawniła tajemnicę oznajmiając, że będzie odchodzić od starego systemu CP/M, dzięki któremu zdobyła swą pozycję rynkową, na rzecz nowego systemu, stanowiącego połączenie cech CP/M i PC-DOS, a zapewniającego eksploatację zastosowań zrealizowanych z użyciem każdego z tych systemów.

Obecnie DRI zajmuje się nie tylko systemem PC-DOS, ale także — dzięki systemowi CONCURRENT DOS — dostarcza wielozadaniowy system operacyjny, który może być również modyfikowany dla zastosowań wielodostępnych. To bardzo ważna możliwość — w świetle aktualnych tendencji rozwojowych komputerów osobistych.

W tej chwili system CONCURRENT DOS jest testowany przez 26 firm. System ten będzie zdolny do wspomagania zastosowań zarówno systemu CP/M, jak i PC-DOS — MS-DOS, eksploatowanych równolegle dla jednego lub wielu użytkowników. Pierwsze dostawy dla ponad 75 użytkowników przewidziano na koniec marca 1984.

Nie mogąc konkurować z systemem MS-DOS firmy MICROSOFT, DRI próbuje wyprzedzić tę firmę przez połączenie CP/M z DOS. Aby to osiągnąć, DRI koncentruje się na ścisłej współpracy z AT&T i IBM. Po półtorarocznym okresie współpracy z IBM, DRI dysponuje obecnie pełną serią kompilatorów i oprogramowaniem narzędziowym, a także standardowym roz-

wiązaniem systemu DOS dla wersji 3270 komputera osobistego IBM PC.

W lutym br. IBM wprowadził do sprzedaży pięć spośród języków programowania DRI i pakiet narzędzi programowania dla PC 3270. Te języki to: COBOL Level 2, PASCAL MT Plus, BASIC C Compiler, C oraz PL/1.

Natomiast przy współpracy z AT&T, DRI zapowiedziała realizację programu wspomagającego system UNIX dla mikroprocesorów 68000 i INTEL 286.

Celem takiej strategii DRI jest osiągnięcie wymienności języków i systemów operacyjnych przez napisanie ich w języku C. DRI dąży do tego, aby sam kompilator, a nie program użytkowy był wymienny dla różnych mikroprocesorów i aby można przenosić zastosowania z jednego systemu na drugi wyłącznie przez rekompilację.

Ze względu na wysoką wydajność procesorów i niski koszt pamięci, DRI rezygnuje z języków symbolicznych, ukierunkowując się na rozwój języków wysokiego poziomu.

W chwili obecnej DRI ma już opracowane języki BASIC, C, FORTRAN, COBOL, PASCAL i PL/1 oraz systemy operacyjne CP/M, CP/M-86 i CONCURRENT CP/M dla mikroprocesorów 8-bitowych 8080 i Z80 a także dla 16-bitowych 8086 i Z8000. Opracowywane są: CONCURRENT DOS, UNIX oraz wersje wspomnianych języków dla kostek 32-bitowych, takich jak 68032 AT&T BELIMAC 32 i prawdopodobnie NATIONAL SEMICONDUCTOR 32032.

Przypuszczalnie ostatecznym wynikiem działań DRI będzie system przenośny, który połączy cechy systemów operacyjnych UNIX i PC-DOS.

Wprawdzie MICROSOFT jest na podobnej drodze, czyniąc pierwszy krok w celu połączenia systemów UNIX i MC-DOS poprzez nadanie swemu systemowi hierarchicznej struktury zbiorów, niemniej DRI — dzięki przenośności swego nowego produktu — nie pozwoli wyrugować się z rynku oprogramowania mikrokomputerów, jak to miało miejsce w roku 1982.

Oprac. MACIEJ ADAMCZYK
Electronics 7, 23 lutego 1984

Prognoza dla informatyki do końca wieku

Można wyróżnić cztery dziedziny techniki komputerowej, w których konkurencja międzynarodowa jest szczególnie intensywna. Są to:

- obwody o dużym stopniu scalenia
- pamięci dyskowe (magnetyczne i optyczne)
- superkomputery
- systemy inteligentne.

Omówione poniżej przewidywania oparte są na raportach dla rządu USA opublikowanych w końcu 1982 i połowie 1983 r.

OBWODY SCALONE

Układy półprzewodnikowe będą rozwijać się w takim tempie jak dotychczas co najmniej przez najbliższe 15 lat. Już od 1978 roku wytwarzane były uniwersalne mikroprocesory 16-bitowe, a w roku 1981 pojawiły się 32-bitowe wielokostkowe jednostki centralne, które wkrótce będą realizowane w jednej kostce. Mikroprocesory 64-bitowe pojawią się w wersji wielokostkowej w roku 1986, by pod koniec dekady również sprowadzić się do jednej kostki. Natomiast mikroprocesory segmentowe, które do roku 1980 wytwarzane były głównie w wersji 4-bitowej, obecnie wytwarzane są z reguły jako 8-bitowe; od roku 1986 zaczną powstawać także w wersji 16-bitowej, a od 1990 — głównie w tej wersji.

Ceny mikroprocesorów będą obniżać się w sposób ciągły. Mikroprocesor 4-bitowy, stosowany w kalkulatorach kieszonkowych, kosztował w roku 1977 6 dol., a obecnie kosztuje mniej niż 2 dol.; w 1987 kosztować będzie 0,85 — 1 dol., a w 1997 — 0,35—0,5 dol. Podobnie mikroprocesor 8-bitowy, którego cena spadła z kilkunastu dolarów w 1977 roku do 3 dol. w 1982, a w 1992 osiągnie 0,7—1 dol., by w 1997 spaść do poziomu 0,5—0,7 dol. Mikroprocesor 16-bitowy, na jakim zbudowane są w większości komputery osobiste, kosztował przed 1978 rokiem ponad 100 dolarów, a obecnie jego cena spadła poniżej 10, by w 1987 r. osiągnąć poziom 2,5—3,5 dol, a dziesięć lat później — 1—2 dol. Mikroprocesory 32-bitowe są jeszcze stosunkowo mało rozpowszechnione i nadal względnie drogie (ok. 100 dol.), ale ze wzrostem produkcji ich cena powinna spaść poniżej 10 dol. w połowie lat dziewięćdziesiątych, kiedy to mikroprocesory 64-bitowe powinny kosztować 20—30 dol.

Obniżka cen nie będzie z czasem tak gwałtowna — ze względu na wzrastające nakłady na wyposażenie produkcyjne. Przemysł elektroniczny jest bowiem jedynym, w którym co pięć lat trzeba gruntownie modernizować większość zakładów produkcyjnych i budować nowe, dwukrotnie droższe.

Tendencję podobną do powyżej opisanej obserwujemy w pamięciach półprzewodnikowych. Koszt kostek pamięciowych będzie malał logartymicznie, ze wzrostem ich wielkości i zagęszczenia. W połowie lat dziewięćdziesiątych kostki pamięciowe powinny kosztować co najmniej o rząd wielkości mniej niż obecnie. Wolne pamięci kosztowały w 1977 roku ok. 0,08 centa/bit, by w 1982 r. obniżyć swą cenę do ok. 0,01, a w 1987 r. do ok. 0,001—0,002 centa/bit. W roku 1992 koszt ten spadnie do 0,0001—0,0005 centa/bit. Cena pamięci o średniej prędkości wynosiła w 1977 r. 0,2, w 1982 — ok. 0,02 centa/bit, a przewidywania na lata 1987 i 1992 dają wartości odpowiednio — 0,003—0,004 oraz 0,0005—0,001 centa/bit. Wreszcie szybkie pamięci, które w 1977 r. kosztowały 0,7 i 0,09 cena/bit, w 1982 r. powinny kosztować w 1987 r. 0,015—0,025, 0,003—0,008 w 1992 r. i 0,0007—0,0035 centa/bit w roku 1997. Zmniejszanie kosztu kostek pamięciowych będzie trwało dłużej niż mikroprocesorów, ze względu na redundacyjne układy logiczne do poprawiania błędów w tych kostkach, które dopuszczają większy procent uszkodzeń produkcyjnych. Jednakże tak jak w mikroprocesorach, kostki pamięciowe o dużym stopniu scalenia i wąskich ścieżkach wymagają na ogół zupełnie nowego wyposażenia produkcyjnego.

Przewidywania obniżki cen wynikają ze stosowania większego stopnia scalenia, zmniejszenia szerokości ścieżek powodującego ograniczenie wielkości poszczególnych elementów, redukcji gęstości uszkodzeń, a także z usprawnień technologii wytwarzania. W pewnym stopniu te same czynniki przyspieszają działanie układu. Opóźnienie na jedną bramkę układu w funkcji czasu dla różnych technologii maleje w różnym stopniu. Dla wolnych układów krzemowych parametr ten wynosił 10 ns w 1977 r. i ok. 3 ns w 1982 r.; przewiduje się, że w 1987 r. spadnie poniżej 1 ns i poniżej 0,2 ns w 1997 r. Szybkie układy krzemowe miały w 1977 r. opóźnienie 0,9 ns, w 1982 spadło ono do 0,35 ns i przewidywany jest dalszy spadek do ok. 0,1 ns w 1990 r. oraz ok. 45 ps w roku 1997. Tak więc zarówno wolne, jak i szybkie układy krzemowe poprawią do 1997 r. swe parametry szybkości o co najmniej rząd wielkości.

Jeśli szybkość ta będzie zbyt mała, np. dla superkomputerów, pojawią się nowe technologie. Układy wykorzystujące zamiast krzemu arsenek galu mają technologię podobną do stosowanej przy układach krzemowych, a dają szybkości pięciokrotnie większe. Układy te stosuje się już w układach telekomunikacyjnych wielkiej częstotliwości, natomiast dla układów cyfrowych opracowano prototypy; nie ulega wątpliwości, że wkrótce nastąpi ich znaczne rozpowszechnienie. W 1977 roku opóźnienie dla tych układów wynosiło 300 ps, w 1982 — ok. 65 ps; na 1987 r. przewiduje się opóźnienie 30

ps, a na 1997 — ok. 17 ps. Konkuruje z nimi technika nadprzewodzących układów kriogenicznych, wykorzystująca złącze Josephsona. Technika ta, opracowana wiele lat temu, napotyka znaczne trudności realizacyjne. Pierwsze układy tego typu miały w 1983 roku opóźnienie 100 ps; przewiduje się, że w 1987 r. uzyskają one mniejsze opóźnienie aniżeli układy GaAs i w 1995 r. przekroczy ono poziom 10 ps. Tak czy inaczej — w 1977 roku powinniśmy z łatwością uzyskiwać szybkości dwudziestokrotnie większe niż obecnie.

Często firmy komputerowe, które dotychczas kupowały elementy, zaczynają je wytwarzać na własne potrzeby, a niekiedy również sprzedawać. Ostatnio WESTERN ELECTRIC i NCR zaczęły sprzedawać kostki zamiast je kupować; czyni to pośrednio również IBM, będąc współwłaścicielem INTEL-LA (w przyszłości może wykorzystywać własne wytwórnie). Poszczególne rządy stosują różne rodzaje pomocy firmom opracowującym układy półprzewodnikowe. Aby jednak utrzymać się w warunkach ostrej konkurencji, wytwórcy muszą ciągle zwiększać inwestycje produkcyjne. Rynek co prawda rośnie, ale nie dość szybko, by zapewnić odpowiedni zysk wszystkim producentom. W konsekwencji prowadzi to do ograniczenia ich liczby.

Coraz bardziej rozpowszechniają się matryce programowane. Są to uniwersalne kostki logiczne zawierające dużą liczbę oddzielnych elementów usytuowanych w rzędach, gdzie dwie lub trzy końcowe operacje wytwórcze zapewniają wzajemne połączenie, wynikające z projektu systemu. W ten sposób projektanci systemów mogą wycofać się z wytwarzania układów półprzewodnikowych bez utraty kontroli nad projektowaniem tych układów. Wspomniane operacje końcowe, pozwalające dostosować matrycę do wymagań użytkownika, są stosunkowo tanie i pozwalają nawet małym firmom, dysponującym odpowiednim zapasem kostek pamięciowych i matrycowych, zachować możliwość projektowania specyficznych układów logicznych.

Można więc przewidywać, że w połowie lat dziewięćdziesiątych istnieć będzie zaledwie kilku potężnych wytwórców takich kostek, przy czym nie jest istotne jakie będą to firmy i w jakim kraju działające. Muszą one osiągnąć najwyższy stopień automatyzacji zapewniający znaczną wydajność, a jednocześnie tak inwestować, by utrzymać ciągły postęp. Osiągane przez te firmy dochody będą w znacznej części przeznaczone na kolejne generacje wyposażenia. Dlatego istotna jest tu rola rządów, które — jak w USA — mogą być gwarantem kapitału inwestycyjnego lub — jak we Francji — właścicielem firm produkujących półprzewodniki.

PAMIĘCI DYSKOWE

Opierając się głównie na cennikach firmy IBM, poczynając od systemu RAMAC 305 w 1955 r. — po system 3380 w 1980 r., można dostrzec jak obniżał się w tym okresie stosunek kosztu pamięci dyskowej do jej pojemności (w centach na jeden przechowywany znak). W 1955 r. parametr ten wynosił niespełna 2 centy, po dziesięciu latach spadł do 0,1 centa, by w 1980 r. osiągnąć 0,01 centa. Przewidywania na dalsze lata wyznaczają 0,0065 centa w 1985 r., a następnie jeszcze gwałtowniejszy spadek — do 0,001 centa w 1990 i poniżej 0,0005 centa w 1995 r.

Obecnie gęstość zapisu informacji na dyskach przekracza już 10 mln bitów/cal² i nadal zwiększa się dzięki cienkowarstwowym głowicom, płaskim ośrodkom zapisu i wielu innych usprawnieniom, przy czym średnica dysku wynosi od 3 cali dla dysków elastycznych do 14 cali dla dysków dużej pojemności typu Winchester.

Przewiduje się, że w najbliższym czasie może tu nastąpić kolejna obniżka kosztu, dzięki dwóm nowym osiągnięciom technicznym: pamięci optycznej i pionowemu zapisowi magnetycznemu. To ostatnie zjawisko polega na usytuowaniu namagnesowanych domen prostopadle do powierzchni podłoża, a nie — jak obecnie — poziomo. W laboratoriach osiągnięto już dla zapisu prostopadłego gęstość 100 tys. bitów/cal² i uważa się, że możliwe jest osiągnięcie gęstości 400 tys. Odpowiada to gęstości powierzchniowej 400 mln bitów/cal², co powinno być osiągane na skalę produkcyjną w roku 1997 i oznacza czterdziestokrotne polepszenie w stosunku do aktualnego poziomu możliwości zapisu. Nie wiadomo, czy będzie to realne — ze względu na wymagania stawiane podzespołom elektromechanicznym przy tak dużych gęstościach zapisu.

Zapis optyczny stwarza szansę na wcześniejsze osiągnięcie gęstości jeszcze większej niż przy pionowym zapisie magnetycznym. Jednakże większość rozwiązań technologii optycznej nie pozwala na ponowny zapis w tym samym miejscu. Jeśli uda się uzyskać zadowalające rozwiązanie z zapisem wielokrotnym lub jeśli zapotrzebowanie na pamięci stałe o dużej pojemności (archiwa, zastosowania biurowe) będzie dostateczne, to produkcja pamięci optycznych powinna wkrótce rozwinąć się na dużą skalę. Wydaje się, że zarówno systemy optyczne, jak i magnetyczne z pionowym zapisem znajdą szerokie zastosowanie.

Postęp w dziedzinie pamięci dyskowych powoduje powstawanie nowych firm. Większość małych dysków stałych oraz dysków elastycznych, które współpracują z komputerami osobistymi i systemami przetwarzania tekstów, opracowano i wyprodukowano w firmach, które dopiero niedawno stały się znaczącymi konkurentami na tym rynku. Aby jednak uzyskać we współzawodnictwie dobre wyniki muszą one stosować wysoce zautomatyzowane procesy, zapewniające osiągnięcie bardzo niskich kosztów wytwarzania dokładnych, a jednocześnie trwałych podzespołów mechanicznych. Tak jak w przypadku półprzewodników, potrzeba tu dużych inwestycji kapitałowych w połączeniu z koncentracją umiejętności technicznych. Aby produkcja była opłacalna, musi ona być naprawdę masowa. Dlatego również tu nastąpi ostra selekcja producentów i dziś trudno przewidzieć, które firmy i w jakich krajach mają największą szansę przetrwania. Obecnie przodują firmy amerykańskie w produkcji zarówno w dużych, jak i małych dysków, ale wytwórcy japońscy ciągle poprawiają parametry swoich wyrobów i wiele z nich eksportuje się już do USA.

Rynek dysków optycznych znajduje się dopiero w stadium rozwoju i wiele firm europejskich prowadzi intensywne prace zmierzające do jak najszybszego rozpoczęcia dostaw i tym samym — uzyskania szansy znacznego rozwoju. Ogólnie można stwierdzić, że pamięci dyskowe uchodzą za inwestycje bardzo rentowne i dlatego łatwo można na nie uzyskać zarówno kapitały prywatne jak i dotacje rządowe na badania.

W latach dziewięćdziesiątych należy spodziewać się ostrej walki konkurencyjnej, w wyniku której powstaną standardy przemysłowe realizowane przez zaledwie kilku producentów jacy pozostaną na polu bitwy.

SUPERKOMPUTERY

Są to procesory o znacznie większej mocy obliczeniowej od tej, jaką mają największe współczesne komputery uniwersalne. Bardziej szczegółowe na ten temat określenia szybko zmieniają się z upływem czasu; np. w 1983 roku za granicę kwalifikują do tej klasy sprzętu przyjmowano 100 mln operacji zmiennego przecinka na sekundę (MFLOPS — million floating point operations per second). Definicja jest tu istotna, ponieważ często myli się to pojęcie z japońskim programem opracowania maszyn piątej generacji, który jest znacznie szerszy, a superkomputery stanowią jeden z końcowych jego etapów.

W lipcu 1982 japońska firma FUJITSU ogłosiła parametry swej maszyny o szybkości 500 MFLOPS. Dwa miesiące później HITACHI podała dane o nowym komputerze uzyskującym 630 MFLOPS. W maju 1983 NEC, trzecia firma japońska, opublikowała parametry wyrobu, który jako pojedynczy procesor liczy z prędkością 700 MFLOPS, a jako system dwuprocesorowy — aż 1300 MFLOPS, a więc szybciej niż przedstawiony nieco wcześniej amerykański komputer CRAY 2 (1000 MFLOPS). Są to jednak tylko zapowiedzi, a nie demonstracje działającego sprzętu. Należy nadmienić, że prędkość obliczeniowa jest ograniczana możliwością podziału rozwiązywanych problemów na elementy dostosowane do jednoczesnego wykonywania obliczeń. Rozwiązanie takie znacznie zwiększa efektywną prędkość, ale trzeba stwierdzić, że wiele istotnych problemów ma z punktu widzenia obliczeń charakter liniowy. Dlatego przy opracowywaniu superkomputerów problemy algorytmów i oprogramowania są ściśle powiązane z rodzajem zastosowanych podzespołów oraz architek-

turą sprzętu. Jest to bardzo prawdopodobne, że doprowadzi to do tworzenia systemów hybrydowych, w których poszczególne części problemu obliczeniowego będą przekształcane na postać równoległą w celu przetwarzania w wyspecjalizowanym procesorze (o prędkości rzędu 100 tys. MFLOPS w roku 1997), natomiast pozostałe części zadania — jak np. operacje wejścia-wyjścia, odsyłanie do zbiorów i przetwarzanie skalarne — realizowane oddzielnie w procesorach funkcjonalnych, dołączonych do szyn danych i sterowania. Cały zespół obliczeniowy jest sterowany przez procesor nadzorujący.

Taki uniwersalny system komputerowy przyszłości mógłby mieć np. postać szeregu połączonych pierścieniowo procesorów — połączonych z sobą szyną sterującą i dołączonych poprzez pamięć wewnętrzną każdego z nich do wspólnej szyny danych. Byłyby to procesory:

— wejścia-wyjścia, dołączony do linii komunikacyjnych, sieci lokalnej oraz wejścia-wyjścia wsadowego

— nadzorujący, dołączony do linii zdalnego sterowania

— wyspecjalizowany, dołączony do zewnętrznej pamięci zbiorów wyspecjalizowanych

— pamięci zewnętrznych, dołączony do stacji dysków magnetycznych i optycznych

— aplikacyjne.

„Procesor wyspecjalizowany" odpowiada tu procesowi równoległemu w superkomputerze. Nazwano go uniwersalnym systemem komputerowym przyszłości, a nie systemem superkomputerowym, ponieważ przewiduje się, że ten typ rozwiązania modularnego pozwoli zrealizować system uniwersalny. W systemie superkomputerowym występowałby „silny" procesor wyspecjalizowany oraz „słabsze" procesory wejścia-wyjścia i zbiorów. Procesory przeznaczone do zastosowań mogą stanowić zespół mikroprocesorów przystosowanych do obliczeń skalarnych i sterowania. Natomiast przy przetwarzaniu wsadowym lub systemach baz danych — parametry procesorów funkcjonalnych byłyby zupełnie inne. Stąd właśnie ów nieco dziwny wniosek, że superkomputery przyszłości będą bardzo podobne do uniwersalnych systemów komputerowych, w których zmieniane będą jedynie rodzaje wykorzystywanych procesorów funkcjonalnych.

Wspomniano już o osiągnięciach japońskich w tej dziedzinie, ale dla utrzymania przodującej pozycji niezbędne są istotne innowacje w architekturze systemów uniwersalnych, algorytmach i oprogramowaniu, gdzie — jak wiadomo — Japończycy mają mniejsze osiągnięcia. Dlatego w końcu lat osiemdziesiątych inicjatywa może wrócić w ręce amerykańskie, a do tego czasu powstaną superkomputery brytyjskie i francuskie. Jeśli więc firmy japońskie chcą liczyć się w dziedzinie superkomputerów w latach dziewięćdziesiątych, muszą nadrobić opóźnienia w dziedzinie oprogramowania.

Modularny charakter superkomputerów i systemów uniwersalnych prowadzi do nowego rodzaju konkurencji. Jeśli protokoły sprzęgu szyn są znane (lub standardowe), to poszczególne firmy mogą specjalizować się w pewnych typach modułów funkcjonalnych.

Większość rządów finansuje badania nad rozwojem superkomputerów ze względu na ich rolę w istotnych dziedzinach życia publicznego. Nie wiadomo jednak, czy rynek superkomputerów będzie dostatecznie duży, by przynieść zyski większej liczbie producentów.

SYSTEMY INTELIGENTNE

Zaproponowany przez Japończyków ogólny schemat komputera piątej generacji składa się z trzech warstw. Pierwszą jest sprzęg zewnętrzny maszyny; możemy w niej wyróżnić język zapytań wysokiego poziomu, dostęp wykorzystujący język naturalny w postaci dźwiękowej i obrazy oraz język dotyczący samego jądra komputera piątej generacji. Warstwa ta łączy się z podstawowym systemem oprogramowania, w którym można wyróżnić odpowiedniki elementów warstwy pierwszej. I tak: językowi zapytań odpowiada system zarządzania „oparty na wiedzy", blokowi dostępu — inteligentny system sprzęgu, a językowi jądra — system rozwiązywania problemów i wyciągania wniosków. Warstwa ta sprzęga się z systemem sprzętowym, gdzie można wyróżnić trzy obszary odpowiadające podziałom w warstwach poprzednich. Część „oparta na wiedzy" obejmuje układy algebry relacyjnej i mechanizmu relacyjnych baz danych. Inteligentnemu systemowi sprzęgu w oprogramowaniu odpowiada taki sam sprzęt, a wśród sprzętu rozwiązującego problemy i wnioskującego można wyróżnić: układy logiczne języka programowania, mechanizm do działania na danych typu abstrakcyjnego, mechanizm przetwarzania przepływu danych i nowatorski mechanizm Von Neumanna. Wszystkie te obszary realizowane są w architekturze bardzo dużego stopnia scalenia. Ze sprzętowej realizacji sprzętu odchodzi też bramkowane wyjście do sieci innych systemów piątej generacji.

Architektura piątej generacji nie jest właściwie architekturą w konwencjonalnym sensie tego słowa. Jest raczej zespołem funkcji, które opracowujący chcieli rozdzielić na nieokreślone jeszcze moduły. Schemat ten może odnosić się do różnych technik. O układach wielkiego stopnia scalenia już była mowa. Poza technikami sprzęgu większość pozostałych związana jest z koncepcjami tzw. „systemów opartych na wiedzy" lub „systemów rozwiązujących problemy". W różnych postaciach systemy te składają się z trzech podstawowych bloków, które łącznie stanowią nowe podejście do rozwiązania złożonych problemów. Te trzy bloki, to:

- **Podstawa wiedzy.** Jest to wiedza na określony temat, reprezentowana zwykle jako warunkowe prawa logiczne, przewidywany rozwój wydarzeń wyrażony w języku naturalnym lub jako wzory matematyczne (lub inne struktury).
- **Dane kontekstowe** — informacje wytworzone przez system o każdej szczególnej sytuacji, w której powstaje program.
- **Mechanizm wnioskowania** — jest to program komputerowy pozwalający wyciągać wnioski i przedstawiać rozwiązania problemu będącego przedmiotem analizy. Programy mają postać „reprezentacji wiedzy" i zawierają ułatwienia do odpowiedniej konwersacji z użytkownikiem. Mechanizm ten działa zwykle w dwóch etapach: wyciąga wnioski o przedstawionej sytuacji, zadając automatycznie pytania użytkownikowi, a następnie prezentuje mu rozumowanie logiczne, na którym opierało się zaproponowane rozwiązanie, podkreślając te momenty, które mogą być użyteczne.

Wszystkie bloki dotyczą oprogramowania i mogą być przygotowane do pracy w konwencjonalnych systemach komputerowych. Jednakże cechy sposobu przetwarzania, potrzebnych pamięci i sterowania sugerują konieczność zastosowania sprzętu specjalnie

Przed sprawdzeniem użyteczności opracowanego do tego celu.

Przed sprawdzeniem użyteczności mechanizmu wnioskowania należy opracować dla niego właściwą podstawę wiedzy i dane kontekstowe. Należy ustalić setki lub nawet tysiące określeń i zależności. Można ocenić, że potrzeba byłoby dwa lata prób jednego człowieka, zanim model systemu rozwiązującego problemy mógłby zacząć działać. Optymiści twierdzą, że system taki będzie użyteczny dla wielu ludzi zajmujących się podobnymi zagadnieniami, natomiast pesymiści uważają, że może służyć tylko autorom. Możliwości takich systemów są bardzo interesujące, ale rozwijać się one będą bardzo wolno i tylko w wybranych dziedzinach. Pojawią się najpierw prawdopodobnie nie jako oddzielny rodzaj systemów, ale zespół opracowań programowych i specjalizowanych procesorów, ukierunkowanych na modularne systemy uniwersalne. W perspektywie systemy rozwiązujące problemy spowodują znaczną, ewolucyjną zmianę w charakterze systemów uniwersalnych.

Dotychczas nie tworzono w tej dziedzinie specjalnych programów badawczych. Sukces odnosiły firmy, które łączyły twórcze podejście w dziedzinie oprogramowania z dobrym rozeznaniem rynku. Większość takich firm powstała w USA i Wielkiej Brytanii. Japońskie zamierzenia stymulują rozwój takich firm na Zachodzie.

Wydaje się, że większość sprzętu przyszłych systemów będzie wytwarzana przez wyspecjalizowanych światowych dostawców modułów, tak jak to już obecnie dzieje się z monitorami ekranowymi i drukarkami. W przyszłości obejmie to matryce programowane do lokalnego przetwarzania, pamięci dyskowe i wyspecjalizowane procesory funkcjonalne. Wytwórcy systemów będą wprawdzie sami produkowali niektóre rodzaje procesorów,

ale nie obejmie to wszystkich modułów żądanych przez klienta.

Już obecnie oprogramowanie jest w dużej mierze dostarczane przez wyspejalizowane firmy. W epoce komputerów osobistych od specjalistów pochodzą nawet programy sterujące, stanowiące dawniej wyłączną domenę wytwórców systemów. Nie wiadomo w jakim stopniu, ze względu na różnice językowe, prawne i zwyczajowe, wytworzy się międzynarodowy rynek oprogramowania. Na pewno jednak przynajmniej programy sterujące systemami oraz automatyzujące niektóre prace inżynierskie (np. CAD — projektowanie wspomagane komputerowo) będą sprzedawane powszechnie przez wspomniane firmy specjalistyczne.

Dostawcy systemów muszą zdobywać i łączyć moduły sprzętu i oprogramowania — tak, aby zaspokoić potrzeby swych klientów. Będą oni obsługiwać organizacje małe i duże, instytucje rządowe i różnego typu szkoły, profesjonalistów i zwykłych konsumentów. Niektórzy będą działać na małą skalę, lokalnie, często w kooperacji — inni będą specjalizować się w określonym typie systemów (np. CAD do projektowania układów półprzewodnikowych), często na skalę światową; jeszcze inni będą tworzyć systemy całościowe spełniające wymagania różnych użytkowników w ramach jednej dużej organizacji.

Struktura przemysłu systemowego i rola, jaką odgrywać w nim będą znani wytwórcy komputerów, może być w przyszłości zupełnie inna. Oprócz wytwórców systemów, ich użytkownicy będą mieli do czynienia z dostawcami usług telekomunikacyjnych. Charakter i koszta dostępnych urządzeń łączności będzie wpływać na wybór systemu, który często sprzedawać będą dostawcy sprzętu łączności. Ważną rolę w przemyśle systemów piątej generacji będą odgrywać rządy. Będą one wspomagać finansowanie badań nad nowymi technologiami, aby wzmocnić pozycję swego kraju. Szczególnie faworyzowane będą przedsięwzięcia perspektywiczne — jak np. systemy inteligentne, gdzie możliwości osiągnięcia zysku są zbyt odległe, by zachęcić inwestorów prywatnych. Natomiast pamięci dyskowe — przykładowo — mają według oceny planistów dostateczny poziom inwestowania i nie potrzebują pomocy rządowej, która skierowana jest głównie na opracowania istotne dla bezpieczeństwa kraju. Dotyczy to najczęściej producentów kostek pamięciowych i matryc programowanych oraz superkomputerów. Jednakże szybkie zmiany w architekturze systemów są przyczyną wątpliwości co do perspektyw firm nadzorowanych przez rządy.

Zakres i wysokość opłat za usługi telekomunikacyjne oferowane przez państwo ograniczą powszechną dostępność systemów. Powstaną jednakże konkurencyjne systemy użytkowe opracowane przez producentów środków łączności lub lokalnych wytwórców systemów komputerowych.

W rezultacie powinny zyskiwać na tym rządy, które często będą chronić własny przemysł — np. przez zniechęcenie do importu i faworyzowanie własnych producentów. Jednakże żaden kraj nie jest w stanie zaoferować pełnego zestawu modułów sprzętowych i oprogramowania. I dlatego rząd, który poważnie ogranicza import, w gruncie rzeczy działa na niekorzyść swych obywateli, uniemożliwiając im korzystanie z nowoczesnych rozwiązań, co w perspektywie może zmniejszyć wydajność i spowodować większe szkody niż doraźne korzyści z protekcjonizmu. Dlatego przemysł informatyczny piątej generacji będzie z natury swej międzynarodowy, a decyzje poszczególnych rządów powinny przyczyniać się do zwiększenia wspólnych korzyści.

Oprac.
JAN RYZKO
na podstawie DATAMATION nr 12, 1983

Reaktywowanie Komitetu NOT ds. Informatyki

Na początku br. odbyło się pod przewodnictwem Zastępcy Sekretarza Generalnego NOT, doc. Tadeusza Skarżyńskiego, zebranie sprawozdawczo-wyborcze poświęcone reaktywowaniu działalności Komitetu Naukowo-Technicznego NOT ds. Informatyki. Na zebraniu przyjęto nowy regulamin Komitetu oraz dokonano wyboru jego nowych władz. W tajnym głosowaniu wybrano Przewodniczącego Komitetu, którym został prof. dr hab. Andrzej Janicki, oraz 8-osobowe Prezydium, w tym trzech wiceprzewodniczących i Sekretarza Komitetu.

Warto przypomnieć, że Komitet jest organem społecznym, a jego statutowymi zadaniami są m.in. funkcje doradcze i opiniodawcze przy opracowywaniu programów nauczania, szkolenia i rozwoju techniki, inicjowanie działalności naukowej i technicznej we wszystkich dziedzinach informatyki oraz popularyzacja jej zastosowań.

Należy wyrazić nadzieję, że wznowienie działalności Komitetu przyczyni się do rozwiązania wielu nabrzmiałych problemów naszego środowiska.

Walor nowej litografii

Firmy elektroniczne na całym świecie muszą poradzić sobie z faktem, że tradycyjne techniki optycznej litografii nie pozwolą wyprodukować następnej generacji mikrokostek.

Najbardziej obiecujące rozwiązanie to litografia elektronowa, w której elektrony zastępują światło w tworzeniu wzoru (ang. pattern) kostki. Wymaga to jednak długiego czasu ekspozycji — z powodu konieczności przeglądania dużych obszarów kostki punkt po punkcie. Dwaj naukowcy z Bell Laboratories w New Jersey zaproponowali ciekawy sposób rozwiązania problemu. Strumień elektronów wykorzystują tylko do tworzenia zarysu wzoru. Swoją metodę nazwali „brush-fire litography" (litografia szybko rozprzestrzeniającego się ognia). Proponowana technika jest dużo szybsza, proces jest mniej niebezpieczny dla podłoża kostki, a uzyskany wzór wyraźniejszy niż przy tradycyjnej obróbce.

Bell Laboratories zaprezentowało swój pomysł na przykładzie tworzenia wzorów na błonie chromowej, umieszczonej na podłożu izolacyjnym. Strumień elektronów rysuje kontur na wrażliwym materiale (błonie ochronnej), którym pokryto powierzchnię chromu. Naświetlony materiał jest usuwany i wytrawia się odsłonięty chrom. Odrzucenie zbędnej warstwy ochronnej pozostawia na kostce wzór zarysowany przez „przesieki" na błonie chromowej, które później ograniczają zasięg reakcji chemicznej. Ich działanie jest analogiczne do roli przesieków w gaszeniu wielkich pożarów — stąd nazwa techniki.

Jeżeli metoda ma być wykorzystana w praktyce, należy określić sposób odróżniania obszarów wzoru, które mają być poddane wytrawianiu, od tych — które trzeba pominąć. Wzory na kostce składają się zwykle z oddzielnych fragmentów wewnątrz ciągłej matrycy. W opisywanym przypadku wszystkie części matrycy mają połączenie elektryczne, podczas gdy fragmenty wzoru są izolowane od matrycy i od siebie nawzajem. Po zanurzeniu błony chromowej w odpowiednim roztworze wytrawiającym, napięcie przyłożone do ciągłej części schematu będzie chroniło ją — gdy izolowane fragmenty będą rozkładane, albo rozkładało — pozostawiając tamte części nietknięte.

Zalety przyspieszenia procesu litografii to nie tylko możliwość zwiększania produkcji kostek. Podczas przepływu strumienia elektronów warstwa ochronna i podłoże rozpraszają część elektronów powodując zacieranie się wzoru. Działanie na dużych obszarach kostki zwiększa nasilenie zjawiska. Problem zostaje jednak rozwiązany dzięki skróceniu czasu ekspozycji.

K.I.

Zasady prenumeraty

Zamówienia i przedpłaty na prenumeratę INFORMATYKI przyjmuje Zakład Kolportażu Wydawnictwa NOT SIGMA. Adres pocztowy: Wydawnictwo NOT SIGMA — Zakład Kolportażu, 00-950 Warszawa, skr. poczt. 1004. Konto bankowe: III O/M NBP Warszawa nr 1036-7490-139-11.

PRENUMERATORZY ZBIOROWI — jednostki gospodarki uspołecznionej, instytucje i organizacje — składają zamówienia w formie wpłaty-przelewu na specjalnym blankiecie opracowanym przez Wydawnictwo. Część tego blankietu zawiera listę tytułów czasopism kolportowanych przez Wydawnictwo.

W przypadku większej liczby odbiorców:
— od 2 do 5 — należy dokonać wpłaty oddzielnie dla każdego odbiorcy
— od 6 odbiorców — dokonać jednej wpłaty, z powołaniem się na znak kancelaryjny pisma przesłanego do Zakładu Kolportażu, a zawierającego wykaz adresów poszczególnych odbiorców i zamawianych dla nich tytułów czasopism oraz numer konta bankowego, z którego dokonany został przelew. Na wpłacie-zamówieniu należy podać łączne liczby egzemplarzy poszczególnych tytułów czasopism.

WPŁATY-ZAMÓWIENIA przyjmowane są w terminach:
- do 28 lutego — na II, III i IV kwartał
- do 31 maja — na II półrocze i IV kwartał
- do 31 sierpnia — na IV kwartał

PRENUMERATA STAŁA-WIELOLETNIA dotyczy tylko prenumeratorów zbiorowych. Zamawiający będzie otrzymywał z Wydawnictwa potwierdzenie kontynuacji prenumeraty wraz z wezwaniem do zapłaty. Zmiany w prenumeracie należy zgłaszać pisemnie w terminach obowiązujących dla składania zamówień-wpłat.

PRENUMERATORZY INDYWIDUALNI — osoby fizyczne — zamawiają prenumeratę dokonując wpłaty na blankiecie opracowanym przez Wydawnictwo NOT SIGMA (dostępnym w urzędach pocztowych) lub na blankiecie przekazu NBP — w terminach j.w. Na odwrocie środkowego odcinka przekazu, przeznaczonego dla adresata-posiadacza rachunku (Wydawnictwa), należy podać tytuł zamawianego czasopisma, okres prenumeraty i liczbę egzemplarzy.

Ceny INFORMATYKI

Cena egz.	Prenumerata w złotych					
	Normalna			Ulgowa		
	kwart.	półrocz.	roczna	kwart.	półrocz.	roczna
100,—	300,—	600,—	1200,—	105,—	210,—	420,—

PRENUMERATA ze zleceniem wysyłki za granicę jest dwukrotnie droższa.

PRENUMERATA ULGOWA. Uprawnieni są do niej: indywidualni członkowie Stowarzyszeń Naukowo-Technicznych, studenci oraz uczniowie szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu przekazu (na odwrocie środkowego odcinka) pieczęcią Koła SNT NOT, wyższej uczelni lub szkoły.

UWAGA!

1. Instytucje zainteresowane prenumeratą naszego czasopisma mogą otrzymać cennik i blankiet-zamówienie w Zakładzie Kolportażu oraz w:
   — Biurach Wydawniczych:
   90-020 Łódź, Pl. Komuny Paryskiej 5a
   50-019 Wrocław, ul. Świerczewskiego 74
   — Oddziale Wydawnictwa:
   40-014 Katowice, ul. Dąbrowskiego 23
   — Oddziałach Wojewódzkich NOT
   — Zarządach Głównych Stowarzyszeń Naukowo-Technicznych
   — Redakcjach czasopism branżowych.
2. Powyższe warunki prenumeraty dotyczą czasopism kolportowanych przez Wydawnictwo NOT SIGMA.

Dodatkowych informacji o prenumeracie udziela Zakład Kolportażu, tel. 40-00-21 wew. 293, 299 lub 40-35-89.

EGZEMPLARZE ARCHIWALNE czasopism wydawanych przez Wydawnictwo NOT SIGMA (od 1977 roku) można zamawiać w Zakładzie Kolportażu, 00-950 Warszawa ul. Mazowiecka 12 (tel. 26-80-16) lub nabywać w Klubie Prasy Technicznej, 00-950 Warszawa, ul. Mazowiecka 12 (tel. 27-43-65).

Sieci łączności w teleinformatyce

Na tle bogatej literatury światowej poświęconej sieciom komputerowym książka ta [1]) odznacza się wysokim poziomem naukowym, zwartym logicznie tokiem wykładu oraz starannością opracowania redakcyjnego. Są to istotne zalety z punktu widzenia kogoś, kto pragnie opanować lub pogłębić podstawy teoretyczne teleinformatyki. W pierwotnym zamierzeniu autora książka ta, o ile mi wiadomo, miała być jedynie nieznacznie zmienionym tłumaczeniem z języka polskiego pracy wydanej w roku 1979 [2]). Znaczny postęp w dziedzinie teleinformatyki na świecie spowodował zapewne, iż autor dokonał w tłumaczonym tekście wielu istotnych zmian i uzupełnień, dzięki czemu można mówić o nowo powstałym dziele naukowym, tematycznie nawiązującym do pierwowzoru, lecz różniącym się od niego i układem wewnętrznym, i w znacznym stopniu — treścią.

Jak wynika z jej tytułu, książka nie obejmuje całości problemów projektowania sieci komputerowych, lecz wyłącznie — podsieci komunikacyjnych sieci komputerowych i to raczej z pominięciem problemów sprzętowych oraz fizycznych zagadnień transmisji sygnałów. Przyjmuje się także, że podsieci komunikacyjne o interesującym autora przeznaczeniu działają na zasadzie komutacji pakietów, bazując na podsieci połączeń fizycznych w zasadzie dowolnej natury. Mając na uwadze znany siedmiowarstwowy układ sterowania procesami w systemach teleinformatycznych, można powiedzieć, że omawiana tu książka obejmuje jedynie trzy najniższe warstwy, a szczególnie — warstwę sterowania pracą pojedynczych kanałów transmisyjnych oraz warstwę sterowania siecią komunikacyjną.

W stosunku do wspomnianej już pracy wydanej w 1979 roku, zostały obecnie pominięte przykładowe opisy sieci komputerowych, wprowadzono natomiast nowy, interesujący rozdział dotyczący przeciążeń w sieciach komputerowych i sposobów zapobiegania im.

Jak zaznaczono w Przedmowie, książka ta jest adresowana do specjalistów z dziedziny projektowania sieci komputerowych oraz do studentów i doktorantów odpowiednich specjalności. Informację tę należy jednak uzupełnić, gdyż książka ta nie może służyć jako pierwsze wprowadzenie do zagadnień teleinformatyki i wymaga od czytelników zarówno znajomości podstaw technicznych transmisji danych, jak i znajomości elementów analizy matematycznej i rachunku prawdopodobieństwa. Znakomitym i w pełni wystarczającym wprowadzeniem do jej studiowania może być inna, niedawno opublikowana praca tegoż autora [3]), wydana w serii podręczników akademickich.

Przy tej okazji warto zwrócić uwagę na ogólnie niski poziom wiedzy w dziedzinie podstaw telekomunikacji absolwentów naszych studiów informatycznych, szczególnie o profilu uniwersyteckim. Stwarza to duże niebezpieczeństwo, że w przyszłej pracy zawodowej nie będą oni w stanie sprostać zadaniom, jakie będzie przed nimi stawiać rozwój systemów teleinformatycznych. Dygresją tą pragnę zwrócić uwagę, iż krąg czytelników omawianej tu książki prof. Jerzego Seidlera, a także książek bliskich jej tematycznie, powinien być szerszy niż wynikałoby to z tytułu akcentującego tematykę telekomunikacyjną.

Nie sposób, oczywiście, w krótkiej recenzji dokonać pełnego przeglądu treści książki i nie jest to moim zamiarem. Pragnę raczej zwrócić uwagę na niektóre jej aspekty metodologiczne. Projektowanie tak złożonych obiektów jak sieci komputerowe przechodzi, jak wiadomo, przez szereg faz, w których znajdują zastosowanie różnorodne metody po-

[1]) Jerzy Seidler: Principles of Computer Communication Network Design. Państwowe Wydawnictwo Naukowe, Warszawa; Ellis Horwood Ltd Publishers, Chichester, 1983

[2]) Jerzy Seidler: Analiza i synteza sieci łączności dla systemów teleinformatycznych. PWN, Warszawa, 1979

[3]) Jerzy Seidler: Nauka o informacji, t. I i II. Podręczniki Akademickie. WNT, Warszawa, 1983

dejścia: od niesformalizowanej pracy koncepcyjnej, poprzez rozważania analityczne, badania symulacyjne — aż do eksperymentalnego sprawdzenia fragmentów systemu lub systemu w całości. Modele analityczne odgrywają w procesie projektowania jednak rolę szczególną: z jednej strony są one bowiem najbardziej precyzyjną formą zapisu efektów wstępnej pracy koncepcyjnej, z drugiej strony natomiast są nieodzowne dla właściwego zaprojektowania eksperymentu symulacyjnego lub badań obiektu fizycznego. Modele analityczne prezentowane w omawianej tu książce dotyczą zarówno pojedynczych kanałów transmisji danych, w tym zwłaszcza kanałów zbiorczych w sieciach terminalowych, jak i rozwiniętych, wielobocznych sieci transmisyjnych, w których (w przypadku komutacji wiadomości lub pakietów) istotnego znaczenia nabiera problem wyboru tras dla połączeń logicznych. Odrębną grupę modeli stanowią wreszcie te, które umożliwiają optymalny wybór konfiguracji połączeń sieci i zdolności przesyłowych kanałów z punktu widzenia potrzeb wynikających z globalnych natężeń ruchu na poszczególnych kierunkach połączeń w sieci.

W tym ostatnim przypadku autor ograniczył się do rozważania jedynie sieci o konfiguracjach drzewiastych. Zubożyło to nieco treść książki, gdyż poza jej polem widzenia pozostały ważne problemy strukturalnej niezawodności sieci komunikacyjnej. Jako podstawowe parametry, charakteryzujące kanały transmisyjne i sieć w rozważanych modelach analitycznych, przyjmuje się przede wszystkim te dotyczące opóźnienia dostawy pakietów do punktu odbioru, prawdopodobieństwo nie dostarczenia pakietu, efektywności wykorzystania zdolności przesyłowej kanału lub sieci itp. Rozważane problemy optymalizacyjne należą w zasadzie do kategorii problemów jednokryterialnych z ograniczeniami, przy czym w książce opisano wiele algorytmów umożliwiających numeryczne rozwiązywanie takich zadań; podano przykłady. Z punktu widzenia programisty pewnym niedostatkiem jest brak informacji o złożoności obliczeniowej algorytmów.

Dla czytelnika zajmującego się nie tylko teorią, lecz także projektowaniem systemów teleinformatycznych, ważne może być przejście od modeli teoretycznych rozważanych w książce do rzeczywistości technicznej. Problemy decyzyjne, które musi rozstrzygać projektant, mogą niekiedy dość znacznie odbiegać od sformułowań zadań optymalizacyjnych wynikających z modeli analitycznych.

Dodatkowe, trudne niekiedy do analitycznego ujęcia, ograniczenia mogą wynikać zarówno z ograniczeń podaży na rynku sprzętowym, jak i z obowiązujących w danym kraju przepisów lub norm technicznych. Rzadko zdarza się, na przykład, by klient zdany na korzystanie z usług telekomunikacyjnych oferowanych przez resort łączności mógł dysponować pełnym spektrum zdolności przesyłowych kanałów, a nie tylko ich znormalizowanymi wiązkami. Podobnie też — z punktu widzenia możliwości wyboru reżymu pracy kanałów transmisji danych — w innej sytuacji jest projektant systemu, w którego dyspozycji znajduje się bazowa podsieć połączeń fizycznych, w innej natomiast ten, który korzysta z sieci dzierżawionej, narzucającej klientom określoną postać protokołów komunikacyjnych niższych rzędów (a więc przede wszystkim — protokołów obsługi kanałów).

Projektant systemu teleinformatycznego w konkretnej sytuacji, mimo bogactwa treści książki, może w niej nie znaleźć modelu teoretycznego dokładnie pasującego do określonej rzeczywistości. Znajdzie w niej jednak coś znacznie ważniejszego: zasady tworzenia modeli analitycznych i metody interpretacji wyników analizy takich modeli w języku inżynierskim. Mówiąc prościej — książka ta uczy, jak stosować metody teorii masowej obsługi, teorii grafów itp. w teleinformatyce. Co więcej, wyniki analizy szeregu modeli, ujawniające ważne zależności funkcyjne między parametrami charakteryzującymi sieć komunikacyjną, wzbogacają naszą wiedzę ogólną, czyli poszerzają bazę tego, co zwykliśmy nazywać intuicją inżynierską, a co w rzeczywistości jest ekstrapolowaniem wiedzy sprawdzonej na innych, zbliżonych modelach rzeczywistości.

Dodatkowym walorem książki Jerzego Seidlera jest to, że przedstawiając aktualny stan wiedzy dotyczącej podsieci komunikacyjnych sieci komputerowych ukazuje na tym tle szereg oryginalnych wyników polskich specjalistów młodszego pokolenia, związanych działalnością z osobą autora. Mamy tu zatem do czynienia z ukształtowaniem się pod kierownictwem naukowym prof. Seidlera szkoły naukowej teorii sieci teleinformatycznych, choć przedstawiciele tej szkoły reprezentują różne instytucje. Można jedynie żałować, że nie pracują oni wspólnie pod jednym kierownictwem i że tak niewiele z ich dorobku wykorzystuje się dotychczas w praktyce. Dorobek naukowy prof. dra inż. Jerzego Seidlera został jednak w 1984 roku wyróżniony indywidualną państwową nagrodą naukową I-go stopnia i wydaje się, że było to zasłużone wyróżnienie.

JULIUSZ LECH KULIKOWSKI

TERMINOLOGIA

Uwagi o polskiej terminologii CHILLA

Oprócz trzyczęściowego cyklu o CHILLU, drukowanego w INFORMATYCE (p. str. 11), opublikowano także dwa inne opracowania w języku polskim, dotyczące tego języka programowania [1]). Ponieważ CHILL jest językiem stosunkowo nowym, nie należy się dziwić, że terminologia stosowana przez autorów tych opracowań nie jest całkowicie jednolita. Istnieją pewne rozbieżności w nazewnictwie, choć na ogół pojęcia CHILLA nie odbiegają zbytnio od występujących od dawna w innych językach. Poniżej przedstawiam własne uwagi dotyczące terminologii używanej w wymienionych publikacjach.

CHILL charakteryzuje się znacznym bogactwem typów danych (po angielsku określa się je słowem **mode**). W większości, są to typy spotykane także w innych językach programowania, choć niekiedy inaczej nazwane. Przykładowo, typ wyliczeniowy nazwano w CHILLU **set mode**, a typ wskaźnikowy — **reference mode**. Jeżeli odpowiednie pojęcie ma już polską nazwę, to autorzy na ogół jej używają, nie szukając nowych odpowiedników nazwy angielskiej, choć w jednym przypadku typ wskaźnikowy nazwano **referencyjnym**. Należy jednak podkreślić, że poszukiwanie nowych polskich nazw dla znanych pojęć jest zbędne, a czasem nawet — szkodliwe.

W niektórych przypadkach istnieją rozbieżności co do nazewnictwa typów, w innych, choć nie ma rozbieżności, można znaleźć trafniejsze odpowiedniki polskie.

W CHILLU wyróżnia się kilka rodzajów typów wskaźnikowych. **Typ związany** (ang. bound), w jednej pracy nazywany **ograniczonym**, określa wskaźniki do zmiennych statycznych i jest kwalifikowany typem zmiennej, którą wskazuje. Dla zmiennych statycznych istnieją też **typy wskaźnikowe swobodne** (ang. free), co do nazwy których panuje zgodność. Typ wskaźnikowy do zmiennych dynamicznych, np. dynamicznych tablic lub rekordów, nazywa się **szeregowym** lub **wierszowym** (ang. row mode).

[1]) Wprowadzenie do CHILLA — języka wysokiego poziomu CCITT. Instytut Łączności, Warszawa, sierpień 1983 (tłum. A. Hildebrandt i. in.;) M. Jarociński, M. Średniawa: Wprowadzenie do języka CHILL, Instytut Telekomunikacji Politechniki Warszawskiej, 1983

Typy różniące się tylko nazwą nazywają się **synonimowymi** (ang. synonym mode). Określenie **typ synonimiczny** wydaje mi się mniej trafne (skoro od rzeczownika **anonim** utworzono przymiotnik **anonimowy**, to dla rzeczownika **synonim** powinno być analogicznie).

W przypadku definiowania nowych typów na podstawie typów istniejących, mówi się o **typach okrojonych** (ang. range mode), nazywając je też po polsku — **zakresowymi**. Wydaje mi się, że bardziej odpowiednim określeniem byłby przymiotnik **zawężony**, jako że typ taki powstaje właśnie wskutek zawężenia (lepsze słowo od — **okrojenia**) zakresu wartości innego typu dyskretnego. Typ, z którego powstaje typ zawężony, należy nazywać raczej **macierzystym** (ang. parent), a nie — **rodzicielskim**, jak w jednej z publikacji.

Pewien kłopot wiąże się ze znalezieniem odpowiednika dla typu zwanego **string mode**, którego wartości mają postać ciągów nazywanych powszechnie, choć niepoprawnie, **łańcuchami**. Ciągi tego rodzaju mogą mieć **typ bitowy** (ang. bit string) lub **znakowy** (ang. character **string**). Słowo **string** stanowi jedynie łączną nazwę dla tych dwóch typów, a ponieważ nie jest słowem kluczowym, można go w ogóle nie używać. Na **podciągach** (ang. substring) lub **wycinkach** (ang. string slice) można wykonywać **operacje konkatenacji** (ang. concatenation) i **wyodrębniania** (ang. substringing), lecz chyba nie — **brania podłańcucha**, jak w jednej z wymienionych publikacji.

Inny kłopot powstaje przy nazwaniu typu INSTANCE, którego obiekty są identyfikatorami procesów. W cytowanych pracach użyto nazw: **typ replikowy**, **egzemplarzowy** i **typ identyfikujący proces**. Wydaje mi się, że nie popełniono by błędu, nazywając ten typ po prostu **typem procesowym**, przez analogię do nazw pozostałych typów i do określenia **typ zadaniowy** (ang. task type), stosowanego w ADZIE (gdzie jednostki programowe wykonywane współbieżnie nazywają się **zadaniami**).

Ponieważ obiekty o typowych właściwościach rekordów nazywa się w CHILLU **strukturami** (ang. structure), na ogół utrzymuje się tę nazwę w języku polskim, choć można mówić również **rekord**. Osobiście głosuję jednak za używaniem słowa **rekord**, jako że struktura w samej informatyce jest obciążona różnymi innymi znaczeniami (choć **rekord** ma podobno w CHILLU także inne znaczenie związane z wejściem-wyjściem). Składowe struktury nazywają się w CHILLU **polami** (ang. field), natomiast obiekty typu strukturowego nazwano **krotkami** (ang. tuple) lub — nie wiem dlaczego — **konstruktorami**. Z dwojga złego, lepszą nazwą jest chyba **krotka**, choć sam chętniej mówiłbym na taki obiekt — **agregat**, podobnie jak w ADZIE. Wybór składowej rekordu, tzn. pola struktury nazywa się **selekcją** (ang. selection).

Specyficznym dla CHILLA pojęciem jest tzw. **lokacja** (ang. location), którą interpretuje się jako pojemnik mogący przechowywać wartości. Deklaracja lokacji przydziela miejsce, w którym może być umieszczona wartość oraz przyporządkowuje jej nazwę i typ. Wydaje mi się, że **lokacja** odpowiada dokładnie zmiennej i nie widzę powodu, ażeby w języku polskim wprowadzać nową nazwę na oznaczenie tego pojęcia. Przeświadczenia tego nabrałem po dość starannym przeanalizowaniu znacznych partii wymienionych prac, w których nie znalazłem ani jednego przypadku, ażeby wyraz **lokacja** nie dał się zastąpić wyrazem zmienna.

Zmienna w CHILLU może mieć szereg atrybutów, czyli właściwości, określonych podczas deklarowania. Jedną z nich jest tzw. **niezapisywalność zmiennych** (ang. read only). Określenie to wydaje mi się trafniejsze od użytej w jednej z prac nazwy **niezmienialność**, ponieważ wartość zmiennej o tym atrybucie może ulegać zmianie (nawet jeśli nie można jej zapisać programowo), np. wskutek zmiany stanu rejestru urządzenia zewnętrznego, oznaczanego przez tę zmienną.

Co do nazewnictwa instrukcji stosowanego przez autorów wymienionych opracowań, mam tylko jedną uwagę. Nie należy tłumaczyć na język polski nazw takich instrukcji, jak IF, CASE czy LOOP. Są to nazwy własne i jako takie nie powinny być nigdy tłumaczone, mimo że wszystkie te słowa angielskie mają swoje odpowiedniki w języku pol-

Czyżo E., Latusek T.: Prekursorzy współczesnej informatyki
INFORMATYKA 1984, nr 11, s. 1

Zwięzła historia wkładu wybitnych matematyków, od Arystotelesa do Johna von Neumana, w stworzenie podstaw informatyki.

Petermann U., Szałas A.: Przerwania — alternatywny mechanizm komunikacji między procesami współbieżnymi
INFORMATYKA 1984, nr 11, s. 4

Charakterystyka nowej metody programowania obliczeń współbieżnych, eliminującej ograniczenia poprzednio stosowanych metod. Omówiono zastosowany prosty język programowania PCI.

Żelazny R.: Narzędzia inżynierii oprogramowania (2)
INFORMATYKA 1984, nr 11, s. 7

Druga część przeglądu współczesnych narzędzi inżynierii oprogramowania. Omówiono dalsze przykłady rozwiązań, w tym również najnowszych, jak metoda HDM oraz HOS.

Udrycki W., Wilczyński W.: CHILL — język programowania systemów komutacyjnych (3). Instrukcje i struktury danych
INFORMATYKA 1984, nr 11, s. 11

Trzecia część charakterystyki języka CHILL. Omówiono funkcje podstawowych instrukcji, struktury danych oraz przykład programu w tym języku.

Чижо Э., Лятусек Т.: Предшественники современной вычислительной техники

Краткая история вклада выдающихся математиков, от Аристотеля до Джона фон Неймана, в создание основ вычислительной техники.
INFORMATYKA 1984, № 11, стр. 1

Петерман У., Шалас А.: Прерывания — альтернативный механизм сообщения между параллельными процессами
INFORMATYKA 1984, № 11, стр. 4

Характеристика нового метода программироввания параллельных вычислений, исключающего ограничения прежде применяемых методов. Обсужден применяемый простой язык программирования PCI.

Желязны Р.: Инструменты технологии производства программного обеспечения (2)
INFORMATYKA 1984, № 11, стр. 7

Вторая часть обзора современных инструментов технологии производства программного обеспечения. Обсуждены дальнейшие примеры решений, в том числе последних, как метод HDM и HOS.

Удрыщки В., Вильчиньскм В.: CHILL язык программирования коммутационных систем (3). Операторы и структуры данных

INFORMATYKA 1984, № 11, стр. 11

Третья часть характеристики языка CHILL. Обсуждены функции основных операторов, структуры данных и пример программы на этом языке.

Czyżo E., Latusek T.: Precursors of contemporary informatics
INFORMATYKA 1984, No. 11, p. 1

Concise history of the contribution of prominent mathematicians, from Aristoteles to John von Neumann, to creation of informatics principles.

Petermann U., Szałas A.: Interrupts — an alternative communication mechanism between concurrent processes
INFORMATYKA 1984, No. 11, p. 4

Characteristics of a new programming method for concurrent computations, which eliminates limitations of the existing methods. Simple PCI language, applied for this purpose, is discussed.

Żelazny R.: Tools for software engineering (2)
INFORMATYKA 1984, No.11, p. 7

Second part of the survey of contemporary tools for software engineering. Further solution examples, including most modern, as HDM and HOS methods, are discussed.

Udrycki W., Wilczyński W.: CHILL — a programming language for commutation systems (3). Statements and data structures
INFORMATYKA 1984, No. 11, p. 11

Third part of the CHILL language characteristics. Statement's functions, data structures and an example of CHILL program are discussed.

Czyżo E., Latusek T.: Vorläufer der zeitgenössischen Datenverarbeitung
INFORMATYKA 1984, Nr. 11, S. 1

Kurze Geschichte des Beitrages von prominenten Mathematikern, von Aristoteles bis John von Neumann, zur Schaffung der EDV-Grundlagen.

Petermann U., Szałas A.: Unterbrechungen — ein alternativer Komunikationsmechanismus zwischen simultanen Prozessen
INFORMATYKA 1984, Nr. 11, S. 4

Eine Charakteristik der neuen Programmierungsmethode für Simultanrechnung, die Begrenzungen der bisherigen Methoden beseitigt. Es wurde die angewendete einfache PCI-Sprache besprochen.

Żelazny R.: Hilfsmittel für Software-Technik (2)
INFORMATYKA 1984, Nr. 11, S. 7

Zweiter Teil einer Übersicht über die modernen Hilfsmittel für Software-Technik. Es wurden weitere Beispiele von Lösungen, einschliesslich der modernsten, wie die HDM- und HOS-Methoden, besprochen.

Udrycki W., Wilczyński W.: CHILL — eine Programmiersprache für Kommutationssysteme (3). Instruktionen und Datenstrukturen
INFORMATYKA 1984, Nr. 11, S. 11

Dritter Teil einer Charakteristik der CHILL-Sprache. Es wurden Funktionen der grundlegenden Instruktionen, Datenstrukturen und ein Beispiel des CHILL-Programmes besprochen.

skim. Odmiennie wygląda sprawa w przypadku, gdy chcemy nazwać pewną klasę instrukcji. Wtedy siłą rzeczy należy je nazwać ogólniej, np. **instrukcje sterujące, instrukcje warunkowe, instrukcje pętli** (lub po prostu — **pętle**) itp. Przestrzeganie tej zasady prowadzi do większej precyzji w wyrażaniu, bo wiadomo wówczas, co mamy na myśli mówiąc, na przykład, **instrukcja EXIT**, a nie wiadomo, gdy mówimy o niej — jak w wymienionych publikacjach — **instrukcja opuszczania, instrukcja zaniechania** itp. Podobnie, **instrukcję SEIZE** nazwano **instrukcją przywłaszczania, dzierżawienia** lub **przechwycenia; instrukcję GRANT — instrukcją udostępniania** lub **przyznawania.** Lecz gdy powiemy wprost **instrukcje SEIZE** i **GRANT**, nie ma wątpliwości o co chodzi.

Aby lepiej zilustrować tę zasadę, warto przypomnieć, że podobnie jest w przypadku typów. Jedna ogólna nazwa, np. **typ całkowitoliczbowy, typ wyliczeniowy** itd., może dotyczyć wielu różnych typów tej samej klasy, opatrzonych nazwami własnymi.

Oprócz nazywania klas instrukcji, można także nazywać czynności, które odpowiadają tym instrukcjom (ale nadal — nie same instrukcje, pozostawiając ich nazwy nie tłumaczone), nawet słowami odbiegającymi od bezpośrednich odpowiedników słów angielskich. Przykładowo, instrukcjom **START, STOP, DELAY** i **CONTINUE**, występującym przy zarządzeniu procesami, odpowiada kolejno **wyzwolenie (utworzenie**, ale nie — **wystartowanie), zakończenie (zatrzymanie), zawieszenie (opóźnienie, wstrzymanie)** i **wznowienie** (a także — **kontynuacja) procesu.**

Poza terminami dotyczącymi struktur danych i wykonywanych w programie działań, w CHILLU można wyróżnić grupę terminów związanych z budową programu. Do tych terminów mam stosunkowo najmniej uwag.

Nazywanie jednej z konstrukcji programowych **regionem** (ang. region) jest niezgodne z abecną tendencją do nazywania jej **rejonem** (por. W. Iszkowski, M. Maniecki, Programowanie współbieżne, WNT, Warszawa, 1980; a także — artykuły w INFORMATYCE), choć może przesadą jest aż taka drobiazgowość z mojej strony.

Inna konstrukcja programowa, służąca do obsługi tzw. **wyjątków** (ang. exception) została nazwana po prostu **handlerem** lub — **programem obsługi wyjątków** (ang. exception handler). Ponieważ trudno bez zastrzeżeń zaaprobować używanie nazwy **handler**, a nie jest to odrębny program, ani nawet moduł, nazwałbym go — **segmentem obsługi.**

Skoro mowa o obsłudze wyjątków, warto dodać, że autorzy wymienionych publikacji nie są zgodni co do tego, jak nazwać po polsku **zgłaszanie wyjątków**, nazywając je **powodowaniem** lub **wymuszaniem wyjątków** (ang. causing exception). Uważam, że nie ma potrzeby wyszukiwania odpowiednika polskiego, ponieważ instrukcja CAUSE wcale nie musi mieć polskiej nazwy, natomiast samą czynność można określić dowolnie, tłumacząc słowo **cause** na język polski lub dobierając inne, według autora bardziej odpowiadające rzeczywistej treści tej czynności.

Ważnymi pojęciami dotyczącymi budowy programu są: tzw. **widoczność nazw** (a nie — **widzialność**, ang. visibility) i **zasięg deklaracji** (a nie — **zakres** ang. scope) oraz — związany z nimi tzw. **okres istnienia obiektu** (ang. life time), rozumiany jako czas, podczas którego obiekt jest obecny w programie, co ma istotny związek z przydziałem pamięci. Bezpośrednie tłumaczenie tego ostatniego terminu angielskiego na **czas życia**, co przyjęto w części cytowanych publikacji, uznałbym za mniej udane.

Pomijając niewiele zresztą błędów terminologicznych, jak na przykład nazwanie, w jednej z prac, **parametrów aktualnych procedury — argumentami (parametr aktualny** nie jest terminem najszczęśliwiej dobranym, ale został już ustalony), a **argumentów operacji — operandami**, trzeba stwierdzić, że wszyscy autorzy podeszli bardzo poważnie do spraw terminologicznych i z dużą starannością dobierali odpowiednie nazwy polskie. Jednakże, należy sądzić, że ostateczne ustalenie terminologii CHILLA może nastąpić dopiero po głębszym wniknięciu w istotę języka i po szerszym jego rozpowszechnieniu.

JANUSZ ZALEWSKI

Z targów do salonu

Zapewne niewielu fachowców zwróciło uwagę na pewną informatyczną ciekawostkę tegorocznych Międzynarodowych Targów Poznańskich. W pawilonie polskiego przemysłu komputerowego, oprócz królującego na środku prawdziwego komputera (a nie atrapy ze światełkami, jak to dawniej bywało), znalazło miejsce także kilka stoisk oferujących oprogramowanie. Maleńkie były to kramiki, oferta — nawet w porównaniu z krajowymi możliwościami — jeszcze mniejsza, niemniej programy pojawiły się wreszcie jako towar!

Złośliwi pewnie zapytają: a ileż to tych programów sprzedano i do jakich krajów? Być może nawet żadnego, przynajmniej na eksport; nie w tym rzecz. Każdy, kto miał do czynienia z dużymi imprezami targowymi, wie, że dla postronnego obserwatora mają one charakter niemal towarzyski. I właśnie na tym polega ich rola: na zawieraniu i odnawianiu znajomości, na bezpośrednim kontakcie — i to nie abstrakcyjnych instytucji, ale konkretnych, reprezentujących je ludzi. Rola targów to także prezentacja ofert — nie po to, by wszystko co wystawione sprzedać w ciągu kilku dni, ale by przyszli kontrahenci mogli uzyskać przegląd sytuacji na rynku w danej dziedzinie.

Z ekspozycją polskiego oprogramowania wiąże się jeszcze jeden, wręcz paradoksalny element — była ona przeznaczona przede wszystkim dla odbiorcy krajowego. Dlaczego więc na Międzynarodowych Targach Poznańskich? A gdzie, jeśli nie tam? Przecież nie ma innego miejsca w Polsce, gdzie można by pokazać pełną ofertę informatyczną.

I tu dochodzimy do sedna. Zarówno z obserwacji MTP, jak i stanu informatyki (że o światowych tendencjach targowych nie wspomnę) wynika, iż potrzebna jest w kraju odrębna, specjalistyczna impreza. I nazwałbym ją raczej salonem, a nie targami — żeby potem nie zawierać na niej dawno przygotowanych kontraktów, mających główny cel w postaci ładnych i budujących komunikatów prasowych. Rzecz w tym, by było stałe miejsce do prezentacji i konfrontacji konkretnych rozwiązań sprzętowych i programowych — a na prawdziwe rozmowy handlowe przyjdzie potem czas w trakcie spotkań roboczych, już poza samą imprezą.

By docenić tę propozycję, wystarczy wiedzieć, że na MTP, w celu obejrzenia polskiej oferty komputerowej trzeba było odwiedzić nie tylko wspomniany pawilon, ale też odnaleźć komputery wśród sukienek, mebli itd. — w ekspozycji polonijnej.

Warto też pamiętać, że brak u nas takiej prasy fachowej, która przedstawiałaby nowości rynkowe. Jeszcze na to za wcześnie. Z salonem natomiast radziłbym się pospieszyć — jeśli zaczniemy już (tzn. w przyszłym roku), nawet wyłącznie w krajowym gronie, to mamy szanse, że za kilka lat drogą konsekwentnego rozwoju będzie to główne spotkanie informatyków z krajów RWPG. A postęp w tej dziedzinie jest tak szybki, że w końcu taki salon musi powstać — nie można tego przegapić.

Oczywiście nie ryzykowałbym urządzenia tej imprezy na terenie MTP — trochę to za duże i za drogie. Na razie może wystarczyłoby miejsca w PKiN w Warszawie (salonie targów książki), w OPT w Katowicach czy choćby w dostatecznie dużej hali sportowej w innym mieście — nie należy tylko zapominać o tym, by miasto to miało odpowiednią bazę hotelową i przyzwoity dojazd z różnych stron kraju. Warto też zwrócić uwagę na panującą dość powszechnie na specjalistycznych salonach zasadę stosowania tzw. dni profesjonalnych, tj. otwierania ich dla szerokiej publiczności tylko przez część ich trwania. Myślę też, że nie można ograniczać się wyłącznie do komputerów uniwersalnych i ich oprogramowania — dziś automatyka przemysłowa, sprzęt medyczny czy poligraficzny to też w dużej mierze komputery.

Na koniec jeszcze ostrzeżenie: w kręgach zbliżonych do Zrzeszenia MERA mówi się o podobnych projektach, ale w formie stałej wystawy. Jest to najlepsza metoda, by utopić pomysł — by wybudować prestiżową, ale nikomu nie potrzebną świątynię. A nie trzeba chyba nikomu tłumaczyć, że czemu innemu służy świątynia, a czemu innemu salon.

MAREK SOBCZYK

Kalendarz 1985

W kalendarzu zapowiadamy tylko te imprezy, o których zostaliśmy poinformowani bezpośrednio przez organizatorów. Podajemy tylko główne informacje — szczegółowe otrzymać można w redakcji.

WIOSNA

- HARDWARE AND SOFTWARE COMPONENTS AND ARCHITECTURES FOR THE 5th GENERATION — Paryż, 5—7 marca, organizator: AFCET INFORMATIQUE
- MIKRONIKA-85, Postęp w budowie precyzyjnego sprzętu elektroniczno-mechanicznego — Warszawa, 15—16 maja, organizatorzy: SIMP, Politechnika Warszawska
- SM 85 — Warszawa, 21—24 maja, organizator: Klub Użytkowników Minikomputerów SM, Polskie Towarzystwo Informatyczne
- PROLAMAT 1985, Software for Discrete Manufacturing — Paryż, 11—13 czerwca, organizator: AFCET

LATO

- IX KRAJOWA KONFERENCJA AUTOMATYKI, Stan i perspektywy automatyki w Polsce — Łódź, koniec czerwca, organizatorzy: Polski Komitet Pomiarów i Automatyki NOT, Politechnika Łódzka
- SIGGRAPH 85, The Twelft Annual Conference on Computer Graphics and Interactive Techniques — San Francisco, 22—26 lipca, organizator: ACM
- WCCE-85, World Conference on Computers in Education — Norfolk, Virginia, USA, 29 lipca—2 sierpnia, organizatorzy: IFIP/TC-3, AFIPS
- EUROMICRO 85, Eleventh Symposium on Microprocessing and Microprogramming; Microcomputers, usage and design — Bruksela, 3—6 września, organizator: EUROMICRO
- ZASTOSOWANIE KOMPUTERÓW W PRZEMYŚLE, IV konferencja naukowo-techniczna — Szczecin, 11—13 września, organizatorzy: OW NOT, Politechnika Szczecińska

JESIEŃ

- VIII MIĘDZYNARODOWE SEMINARIUM NT. SYSTEMÓW ZARZĄDZANIA BAZĄ DANYCH — Pieštany (Czechosłowacja), 30 września — 4 października, organizator: VUSEI-AR, Bratysława

Amatorzy mikroinformatyki, uwaga! Podejmujemy starania o wydzielenie mikroKLANU z INFORMATYKI i powołanie osobnego miesięcznika. Oba pisma robione byłyby przez ten sam zespół redakcyjny. Prosimy o listy — jak powinno Waszym zdaniem wyglądać takie czasopismo. Pojawi się ono nie wcześniej niż w połowie przyszłego roku.

Pojawiła się możliwość wydawania przez naszą redakcję serii niewielkich książek związanych z informatyką. Zaczynamy od tematów mikroinformatycznych: wprowadzenie do mikroinformatyki, konstrukcje mikrokomputerów, opisy mikroprocesorów, zbiory programów użytkowych. Liczymy na Waszą pomoc w wyborze tematów i ich realizacji.

Czwartą stronę okładki wyjątkowo poświęcamy na informacje czy ogłoszenia. Jest ona przeznaczona na reprodukcje grafiki komputerowej. Trwa nieustający konkurs na najciekawsze realizacje. Nagroda na razie skromniutka: publikacja (wraz z opisem idei) i honorarium. W przyszłości znajdzie się może większa zachęta. Liczymy na prace oryginalne, nie naśladujące twórczości plastyka — w atrakcyjny sposób wykorzystujące możliwości komputera.

Od stycznia 1985 obowiązują następujące ceny ogłoszeń publikowanych na naszych łamach:
- *ogłoszenia duże (zależnie od objętości):*

cała strona — 35 tys. zł, 3/4 — 30 tys., 1/2 — 25 tys., 1/4 — 20 tys., 1/8 — 15 tys.
- *ogłoszenia drobne: (zależnie od liczby słów):*

jedno słowo — 30 zł.

Dodatki do ceny podstawowej:
— *za dodatkowy kolor (na okładce) +30%*
— *za zamieszczenie ogłoszenia na czwartej stronie okładki +100%*
— *za zamieszczenie ogłoszenia na trzeciej stronie okładki +50%.*

Zniżki:
— *za ogłoszenie 3—5-krotne —5%*
— *za ogłoszenie 6—10-krotne —10%*
— *za ogłoszenie 11-krotne i powyżej —20%.*

Prosimy o niezwlekanie z przekazywaniem nam tekstów ogołszeń. Bardzo ograniczona objętość pisma może potem utrudnić opublikowanie ich we właściwym czasie.

Bogatsze instytucje mogą się pokusić o wykupienie na naszych łamach stałych rubryk. Zniżki w stosunku do podstawowych cen ogłoszeń są wówczas następujące:
— *za artykuły reklamowe i wkładki wykonane przez zleceniodawcę —40%*
— *za bloki i biuletyny wykonane przez zleceniodawcę — maks. 60%.*

Umawiamy się więc na opublikowanie konkretnej liczby kolumn (strona w piśmie) w roku — instytucja ma je do dyspozycji. Nasza rola ogranicza się wtedy do redakcyjnych poprawek. Za merytoryczną stronę artykułów odpowiada zleceniodawca.

Prosimy o listy z oceną dotychczasowego poziomu INFORMATYKI. Pomogą nam przy wszystkich reorganizacjach, które zmienią profil i podniosą — mamy nadzieję — poziom pisma. Prosimy o zaznaczenie czy list może być opublikowany.