Witold Komorowski

# UKŁADOWA IMPLEMENTACJA JĘZYKÓW WYSOKIEGO POZIOMU

Wprowadzenie przez Sun Microsystems mikroprocesora picoJava pod hasłem "*casting Java in silicon*" zaktualizowało problem wpływu języków programowania na architekturę procesorów. Wpływ odwrotny – budowy procesora na sposób jego programowania, można obserwować od zarania komputerów a i obecnie bywa widoczny nawet na poziomie systemu operacyjnego, czego przykładem jest MS DOS z wymuszoną segmentacją pamięci i różnorodnymi zawiłymi próbami jej rozszerzenia (*extended, expanded*). Wiele współczesnych maszyn nosi w sobie ślady poprzednich, często prymitywnych, wcieleń, które przetrwały czasem w postaci szczątkowej dzięki dążeniu producentów do zachowania kompatybilności i w efekcie składają się na eklektyczną architekturę szczególnie dokuczliwą na przykład przy programowaniu w asemblerze. W ciągu półwiecza rozwoju komputerów niewiele było prób konstruowania maszyn z wyraźnie sformułowanym zadaniem dostosowania ich organizacji logicznej do określonego języka. Dominująca obecnie koncepcja RISC [8,18] przyjmuje nawet przeciwne założenie: skonstruować procesor, być może niewygodny do programowania, ale szybki – kompilator i tak przesłoni przed użytkownikiem wszelkie niedostatki architektury.

## Początki

W pierwszych maszynach, sprawa ich programowania nie była traktowana jako istotny problem. Słynny ENIAC – pierwsza elektroniczna maszyna cyfrowa, która powstała jako jednostkowe urządzenie, nie była wedle obecnych kryteriów komputerem (mimo, że ten termin pojawia się w jej nazwie - *Electronic Numerator Integrator And Computer*). Nie spełniała podstawowego kryterium odróżniającego komputery od innych maszyn - nie przechowywała programu swego działania w pamięci. Maszyna ta w ogóle nie była programowana w dzisiejszym tego słowa znaczeniu, choć słowo "programowanie" pojawia się pierwszy raz właśnie w jej opisie [7], lecz raczej ustawiana do określonego typu obliczeń. Jak większość nowatorskich produktów techniki, również pierwsza maszyna cyfrowa zawierała analogie do swoich

poprzedników, w tym przypadku kalkulatorów mechanicznych, np. cyfra dziesiętna była reprezentowana przez rejestr 10-pozycyjny, w którym zawsze tylko jedna pozycja jest wyróżniona przesuwając się cyklicznie na wzór kółka zębatego w liczniku.

Te skojarzenia mechanistyczne długo przechowały się w terminologii: najstarsze na świecie, 40-letnie, stowarzyszenie informatyczne do dziś nosi nazwę *Association for Computing Machinery*, w Polsce, pionierską instytucją był Zakład Aparatów Matematycznych, jego następca nosił nazwę Instytutu Maszyn Matematycznych, na uczelniach istniały katedry Konstrukcji Maszyn Cyfrowych itd. *Nota bene*, w polszczyźnie dość późno pojawiło się słowo “komputer” wprowadzone po kampanii prasowej z użyciem prowokacyjnie ośmieszających propozycji spolszczenia, takich jak “liczółka” lub “samorach”.

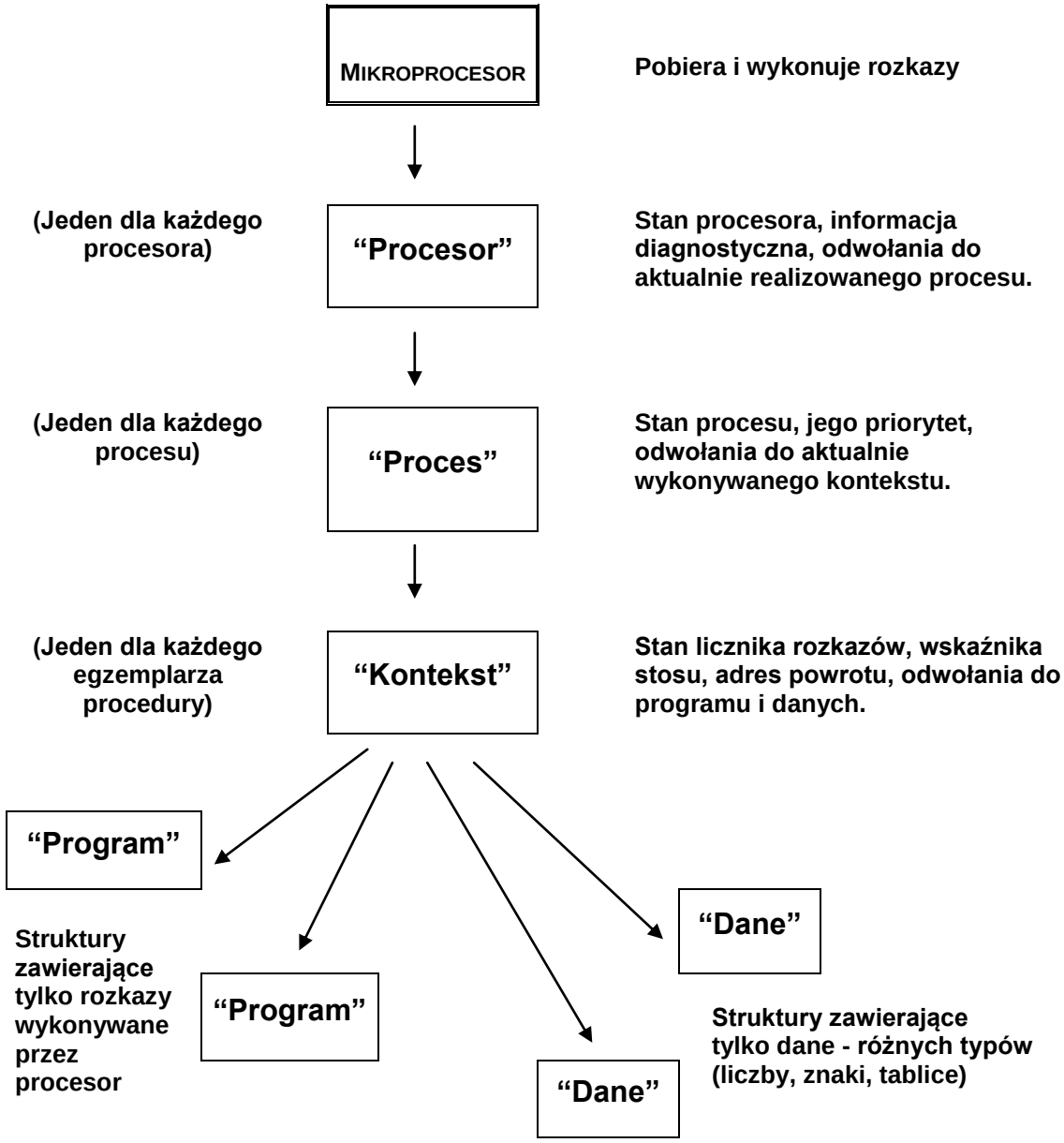
Komputery pierwszej generacji budowane były przy założeniu, że to sami użytkownicy będą je programować a główne wysiłki konstruktorów były skupione na zapewnieniu coraz większej szybkości obliczeń i niezawodności działania, odsuwając wszelkie ułatwienia obsługi i programowania na dalszy plan. Ciekawym przykładem dominacji architektury nad względami użytkowymi jest pierwszy produkowany przemysłowo w Polsce komputer UMC 1. Był on zaprojektowany na Politechnice Warszawskiej i w latach 1962-1964 produkowany we Wrocławskich Zakładach Elektronicznych Elwro (25 egzemplarzy, z tego jeden – wyeksportowany na Węgry) [4]. UMC 1 miał słowo 36-bitowe, pamięć (bębnową) o pojemności 4k słów, wykonywał 100 dodawań na sekundę a wyróżniał się wielu oryginalnymi rozwiązaniami takimi jak np. unikatowy system kodowania liczb z podstawą systemu równą -2. Była to maszyna szeregowa (przetwarzanie bit po bicie) i mikroprogramowana poziomo, przy czym mikroprogramowanie było dostępne dla użytkownika-programisty [12]. Oznacza to, że każdy rozkaz należało skomponować z 22 bitów mikrooperacji tak, aby uzyskać właściwy przepływ międzyrejestrowy. W stosowanym w UMC 1 języku symbolicznym W20, lista rozkazów była podzbiorem teoretycznie możliwych ponad 4 milionów rozkazów.

Prawdopodobnie pierwszym komputerem zaprojektowanym jako system sprzętowo-programowy i uwzględniającym fakt, że będzie programowany w konkretnym języku wysokiego poziomu (Algol), była maszyna B 5000 (architektura

implementowana w komputerach firmy Burroughs produkowanych w latach 1963 – 1968) [1,10]. Była to maszyna o organizacji stosowej, 0-adresowej - alternatywnej dla powszechnej wówczas organizacji 1-adresowej z rejestrami uniwersalnymi. Stos zrealizowano w pamięci operacyjnej ale jego wierzchołek (dwie najwyższe pozycje) umieszczony był w rejestrach procesora i ich zawartość stanowiła domyślne argumenty operacji. Stos ten był używany zarówno jako miejsce składowania argumentów i wyników operacji (stos obliczeniowy), jak i do przechowywania adresów przy wywoływaniu procedur oraz obsłudze przerwań (stos powrotów). Architektura B 5000 nie znalazła bezpośrednich następców i w latach 60. i 70. dominują komputery 1- i 2-adresowe z rejestrami uniwersalnymi stosowanymi jako akumulatory i/lub rejestry adresowe (IBM S/360, PDP-11).

## iAPX 432

Najbardziej ambitnym przedsięwzięciem zmierzającym do zaprojektowania architektury przystosowanej do języka programowania wysokiego poziomu, był mikroprocesor iAPX432 firmy Intel przeznaczony dla języka Ada [16,17]. Był to 32-bitowy mikroprogramowany mikroprocesor zawierający pierwotnie (w chwili wprowadzenia w r. 1980) 3 moduły: układ sterowania i układ wykonywania rozkazów łącznie tworzące procesor (GDP – *general data processor*) oraz układ sprzęgu (IP – *interface processor*). Następnie, w r. 1983 wprowadzono jeszcze 2 moduły: układ sterowania magistrali (BIU) i układ sterowania pamięci (MCU). Procesor umożliwiał bezpośrednie tłumaczenie wielu instrukcji Ady na pojedyncze rozkazy i432. Rozkazy te miały złożoną strukturę i były kodowane jako łańcuchy binarne o zmiennej długości – od 6 do 321 bitów. Wszystkie argumenty mogły być umieszczone wyłącznie w pamięci lub na stosie obliczeniowym. Architektura była ukierunkowana na operowanie raczej “obiektami” niż słowami czy bajtami. Wyróżniono 5 typów obiektów: dane, programy, konteksty, procesy i procesory. Najprostszy z tych obiektów - “dane”, mógł zawierać liczby 16-, 32-, 64- a nawet 80-bitowe (format zmiennoprzecinkowy).




**Hierarchia obiektów w procesorze i432.**

Projekt procesora, podobnie jak i samego języka Ada, był sponsorowany przez Dep. Obrony USA i miał dostarczyć narzędzia do budowy dużych systemów militarnych działających w czasie rzeczywistym. Wynikały stąd wymagania

- łatwości programowania w języku wysokiego poziomu (Ada) jako języku rodzimym,
- niezawodności zarówno sprzętu jak i oprogramowania,
- łatwości pielęgnacji,
- łatwości rozbudowy ,
- możliwości tworzenia systemów wieloprocesorowych.

W konstrukcji i432 zastosowano szczególne sposoby zwiększenia niezawodności – również na poziomie sprzętu. Np. moduły mogły pracować w tzw. trybie "*master-checker"* polegającym na łączeniu 2 układów pracujących równolegle, z tymi samymi sygnałami wejściowymi i wykonujących identyczne działania z sygnalizacją ewentualnych niezgodności. W systemach wieloprocesorowych mikroprocesory komunikowały się przez wspólny obszar pamięci (*interprocessor message area*) stosując pakietowy protokół połączeniowy niezależny od struktury układu. Pakiety były różnej długości i miały różne funkcje, np. pakiet mógł przenosić komendy systemu operacyjnego. Procesor i432 miał wiele funkcji systemu operacyjnego zrealizowanych sprzętowo; "*silicon operating system*" zajmował większość pamięci stałej - zaledwie 6% mikrokodu realizowało podstawowe rozkazy procesora a dodatkowe 7% służyło do organizacji adresacji wirtualnej.

Komplikacja architektury, komunikacja na zasadzie wymiany pakietów i rozbudowane funkcje zwiększające niezawodność spowodowały, że i432 był stosunkowo wolny (wolniejszy nawet od 80286), choć i tak miał w zastosowaniach obliczeniowych większą wydajność niż np. IBM 370/148, co uzasadniało określanie go jako *micromainframe*.

Eliminacja w i432 rejestrów programowych była przeciwieństwem powstającej w tym samym czasie koncepcji RISC (RISC I – Uniw. Berkeley, MIPS – Uniw. Stanforda, 801 - IBM). W komputerach tych (*Reduced Instruction-Set Computers*) rozbudowywano zestaw rejestrów procesora (np. 138 rejestrów w RISC I) ograniczając kontakty z pamięcią do dwóch przesłań – ładowanie (*load*) i zapamiętanie (*store*) rejestru. Uważa się niekiedy, że opracowany później procesor i960 jest RISCową wersją 432.

## Forth

Język Forth, choć powszechnie mało znany, jest do dziś nie tylko stosowany ale i rozwijany w swojej “niszy aplikacyjnej” jaką stanowią mikroprocesory wbudowane (*embedded)*. Według anegdoty, Charles (Chuck) Moore projektując ten język w latach 70. dysponował systemem, który zezwalał na stosowanie nazw tylko 5-literowych i to pisanych dużymi literami – stąd, uważając FORTH za język 4. generacji (*fourth generation*), po usunięciu środkowego “u” , pozostawił obecną nazwę. Język ten ma wiele dialektów i doczekał się chyba największej liczby implementacji układowych, a nawet dał asumpt do powstania pewnej koncepcji architektury zwanej MISC.

Popularność Forth, wynika z faktu, że jest on narzędziem programowania zarówno aplikacji jak i prostych systemów operacyjnych, nie wymaga dużej pamięci i daje łatwe do uruchamiania programy. Mimo, że jest językiem wysokiego poziomu, umożliwia penetrowanie struktury systemu z dokładnością zbliżoną do asemblera – co jest istotne przy programowaniu i uruchamianiu systemów sprzętowo-programowych. W systemach takich procesor stanowi tylko część całości urządzenia i musi przetwarzać sygnały napływające w tempie narzuconym przez proces zewnętrzny, a sam program i związany z nim system operacyjny powinny być możliwie niewielkie aby je umieścić w pamięci stałej wbudowanego mikrosterownika.

Program w tym języku składa się z tzw. “słów” (operacji) przechowywanych w “słowniku”. Cechą Forth jest jego otwartość polegająca na możliwości uzupełniania słownika nowymi słowami tworzonymi przez programistę, czego konsekwencją programową jest konieczność częstego wywoływania na ogół krótkich podprogramów. Fakt ten znalazł swój wyraz w modelu języka (maszynie wirtualnej) bazującym na dwóch stosach: stosie obliczeniowym – dla argumentów i wyników operacji, oraz tzw. stosie powrotów – przeznaczonym głównie dla adresów powrotnych - po wywołaniu podprogramu.

Pierwszą implementacją sprzętową maszyny wirtualnej Forth był 16-bitowy mikroprocesor NC4000 (NOVIX) dostępny w 1985 [9,11,14]. Procesor ten, zawierający tylko 4000 bramek, dzięki specyficznej organizacji dostępu do stosów i pamięci wykonywał więcej niż jedną operację w jednym cyklu zegara, co zapewniało

dużą szybkość przetwarzania. W strukturze procesora są 2 stosy zrealizowane jako szybkie pamięci zewnętrzne o pojemności 256 słów 16-bitowych każda:

- stos adresów, dostępny przez port R – do realizacji wywołań procedur, podstawowego mechanizmu programowania Forth,
- stos danych, dostępny przez port S – do obliczeń.

Na wierzchu stosu danych są dodatkowo 2 rejestry: Top i Next używane w operacjach dwuargumentowych wg schematu

T ←T **op** N

Pamięć ma 64K słowa przy adresowaniu 16-bitowym lub 2M słowa przy rozszerzeniu adresów o 5 bitów rejestru X spełniającego poza tym rolę portu we-wy. Dzięki osobnym szynom łączącym procesor z każdym ze stosów, z pamięcią i z portami we-wy możliwe jest zwielokrotnienie wykonywania rozkazów w czasie (np. łączenie funkcji pobrania argumentu i wykonania operacji arytmetycznej bądź logicznej, wykonywanie działań na stosie bez utraty drugiego argumentu itp.).

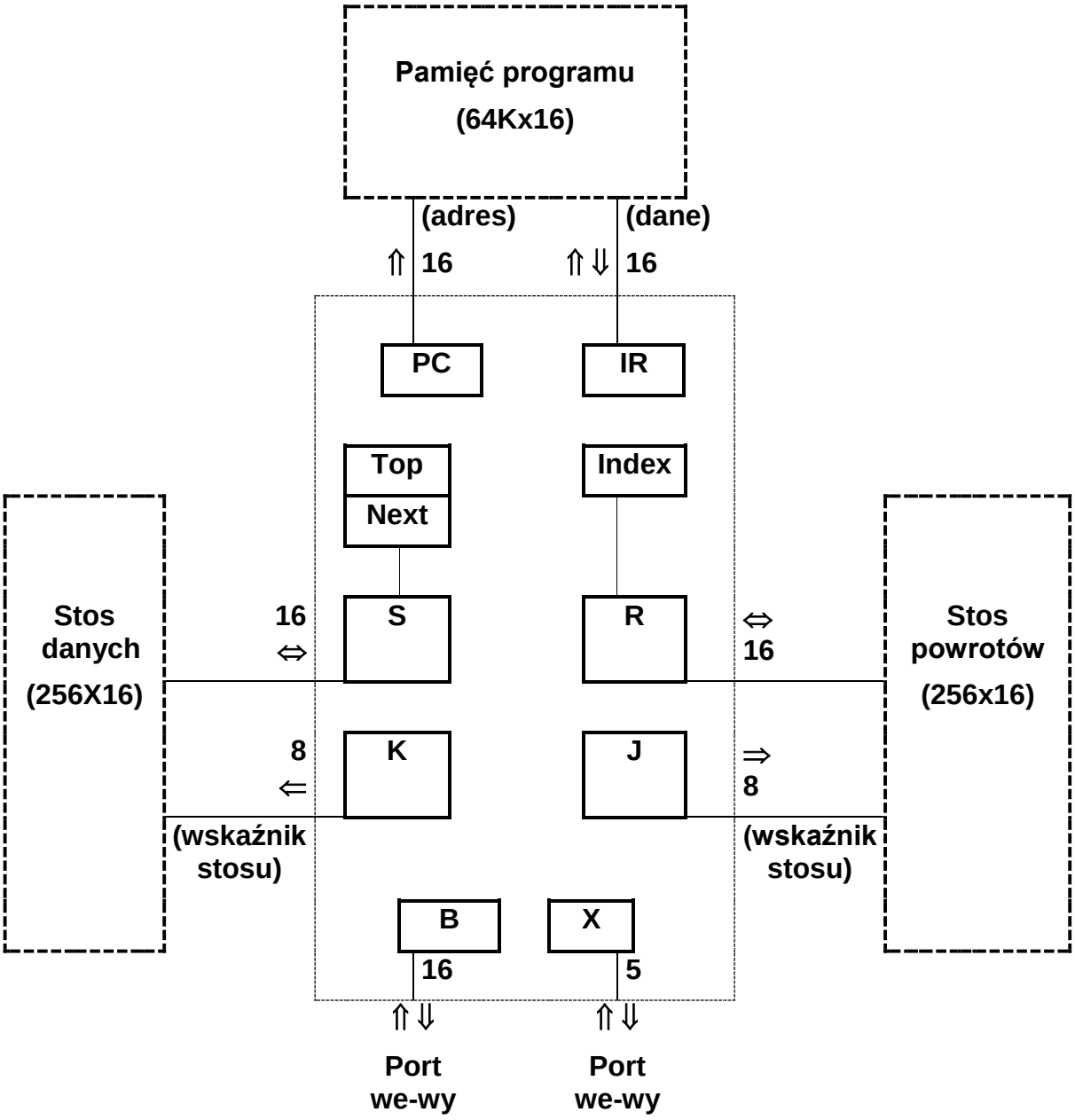
Ewolucja procesorów języka Forth doprowadziła do sformułowania przez Ch. Moore'a założeń architektury MISC (*Minimal Instruction Set Computers*) jako optymalnej dla zastosowań w mikroprocesorach wbudowanych. Architektura MISC pozwala na tanią realizację szybko działającego układu VLSI unikając złożonych układów sterowania charakterystycznych dla CISC jak też – z drugiej strony - specjalnych zabiegów przyspieszających działanie koniecznych w procesorach RISC (potokowość, przewidywanie skoków, szybkie pamięci cache).

Typowym przedstawicielem MISC jest procesor MuP21 (Offete Enterprises, Inc.). Jest to 20-bitowy procesor o architekturze dwustosowej (jak Novix) bez rejestrów uniwersalnych; wszystkie działania wykonują się na zawartości rejestru T, a ewentualny drugi argument pobierany jest ze stosu. Procesor jest przystosowany do programowania w języku Forth i ma tylko 24 rozkazy: 5 rozkazów sterujących - JUMP, CALL, RET, JZ, JCZ; 7 rozkazów arytmetycznych i logicznych: ADD, ADDNZ, AND, XOR, COM, SHL, SHR (nie ma rozkazu odejmowania ani rozkazu sumy logicznej) i 11 rozkazów przesłań pomiędzy pamięcią, stosem i rejestrami adresowymi. Wszystkie rozkazy są 5-bitowe, pakowane po 4 w słowie. W fazie pobrania słowo adresowane przez licznik rozkazów jest przesyłane do rejestru

rozkazowego, skąd kolejne rozkazy są dekodowane dając naturalny efekt potokowości. Po wykonaniu 4 rozkazów następuje odczytanie następnego słowa z pamięci, która może być 4-krotnie wolniejsza niż procesor i dzięki temu nie jest potrzebne stosowanie kosztownej pamięci cache. Zastosowano tu w miniaturze podobną zasadę jak w architekturze VLIW (*Very Long Instruction Word*) – parafrazując, można by ją scharakteryzować jako “niezbyt długie słowo rozkazowe” (prototypowy IBMowski procesor VLIW ma słowo o długości 759 bitów i znacznie bardziej wyrafinowane metody kompletowania tego słowa przed wykonaniem).

Procesor MuP21 współpracuje z koprocesorem sterującym pamięcią i koprocesorem sterującym standardowym monitorem TV przewidywanym jako podstawowe urządzenie wyjściowe w zastosowaniach takich jak zaawansowane gry video, systemy CAD czy komputery podręczne (*handheld*). MuP21 mieści się w obudowie DIP z 40 wyprowadzeniami (co było m.in. powodem ograniczenia wielkości słowa do 20 bitów), zawiera tylko ok. 7000 tranzystorów i wykonuje rozkazy w cyklu 10 ns przy poborze mocy 50 mW. Ze względu na stosowaną wolniejszą pamięć DRAM efektywna szybkość wynosi tylko ok. 80 MIPS.

Procesory języka Forth stanowią również jądro systemów TDS 2020 i 9092 (Triangle Digital Services Ltd.) z mikroprocesorami Hitachi 6301 (8-bitowy) i H8/532 (16-bitowy). Architekturę MISC mają procesory F21 (Ultra Technology Inc.) oraz i21 (iTV Corporation).

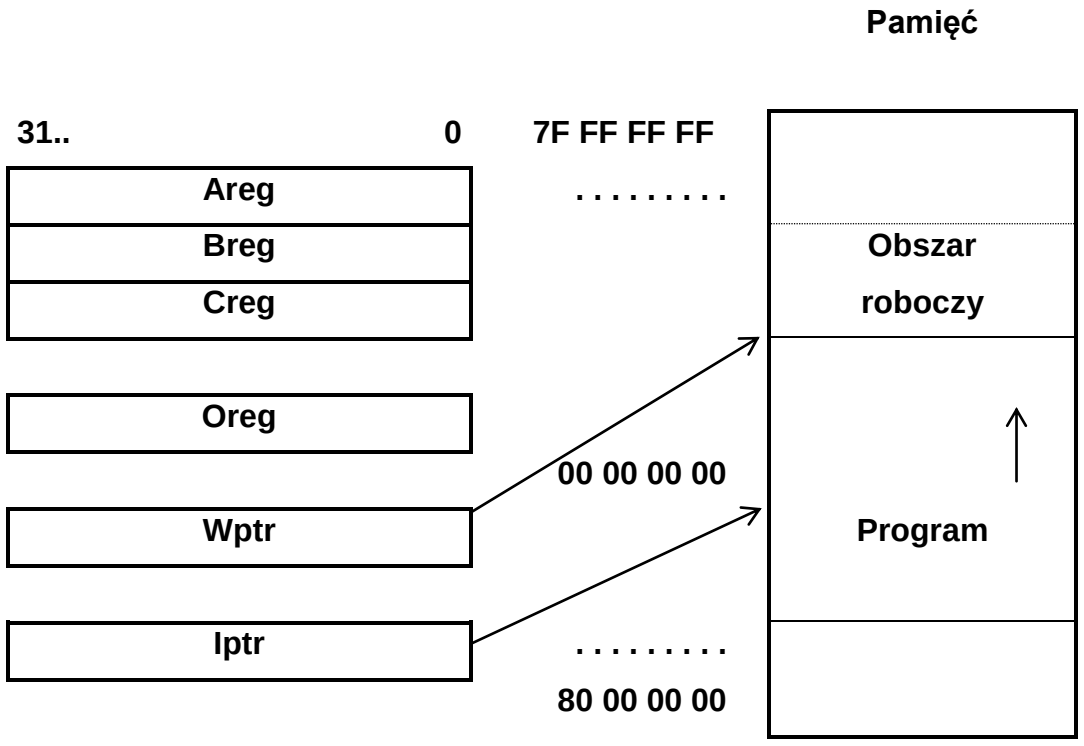



**Struktura procesora NC4000 (NOVIX)**

## Transputer i Occam

Jedynym projektem zawierającym spójną koncepcję procesora i języka jest, opracowany przez firmę Inmos, Inc., transputer i język Occam przeznaczone do przetwarzania równoległego (1983). Program w języku Occam, opisujący kilka procesów współbieżnych może być wykonywany na jednym transputerze – przez podział czasu między procesy, lub na sieci połączonych transputerów [13,14].

Transputer jest układem scalonym VLSI zawierającym 32-bitowy procesor typu RISC, 4 KB pamięci RAM i 4 dwukierunkowe kanały we-wy służące do komunikacji z innymi transputerami tworzącymi sieć. Pierwszy model transputera – T414 (dostępny w 1985) został zastąpiony szybszą wersją T800, a później modelem T9000 wyposażonym w jednostkę zmiennopozycyjną o wydajności 20 Mflops, 16 KB pamięci i kanały o przepustowości 100 Mb/s.




**Podstawowe rejestry transputera.**

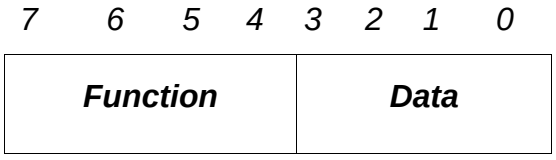
Wszystkie rejestry transputera są 32-bitowe. Rejestry Areg, Breg i Creg tworzą stos obliczeniowy używany do przechowywania argumentów operacji. Rejestr Iptr spełnia rolę licznika rozkazów wskazując adres 8-bitowego rozkazu, który ma być wykonany. Rejestr Oreg przechowuje argument rozkazu; w niektórych przypadkach zawartość Oreg stanowi rozszerzenie kodu operacji. W fazie pobrania rozkazu z pamięci, 4 mniej znaczące bity rozkazu (pole Data) są zawsze przesyłane do Oreg. Rejestr Wptr wskazuje adres początkowy (bazę) obszaru roboczego zawierającego zmienne lokalne aktywnego procesu. Adres w Wptr dotyczy 32-bitowego słowa.

Procesor utrzymuje dwie kolejki procesów (zdefiniowanych przez ich obszary robocze) o wyższym i niższym priorytecie; początki i końce tych kolejek pokazują 2 pary rejestrów. Z każdym priorytetem związane są ponadto 3 rejestry obsługi zdarzeń uwarunkowanych czasowo:

- ClockReg_ - rejestr zegara,
- TptrLoc_ - rejestr wskazujący pierwszą pozycję w kolejce procesów zawieszonych ze względu na czas,
- TnextReg_ - rejestr wskazujący czas następnego zdarzenia.

Dwa jednobitowe znaczniki Tenabled_ wskazują czy odpowiednie kolejki czasowe nie są puste. Transputer nie ma konwencjonalnych znaczników warunków; skoki warunkowe badają stan rejestru Areg.

Podstawowy format rozkazu jest 8-bitowy - cztery bardziej znaczące bity rozkazu zawierają kod operacji (*Function*), pozostałe 4 bity stanowią argument (*Data*).




**Transputer. Format rozkazu.**

Transputer (w wersji T414) miał listę ok. 90 rozkazów, z których 16 stanowi listę podstawową rozkazów kodowanych w jednym bajcie. Ocenia się, że rozkazy podstawowe stanowią ok. 80% rozkazów wykonywanych w typowych programach; wzgląd na częstotliwość występowania był kryterium doboru kodów operacyjnych. W formacie tym są 3 rozkazy specjalne: *opr (Operate), pfix (Prefix) i nfix (Negative*

*prefix).* Rozkazy *pfix* i *nfix* służą do przygotowania zawartości rejestru Oreg; rozkazy te, w przeciwieństwie do wszystkich innych, po wykonaniu nie zerują Oreg. Rozkaz *pfix* wpisuje swoje pole *Data* na 4 mn.zn. bity Oreg i przesuwa zawartość tego rejestru o 4 pozycje w lewo. Rozkaz *nfix* działa analogicznie, lecz przed przesunięciem Oreg uzupełnia (uzupełnienie logiczne) jego zawartość. Rozkaz *Operate* umożliwia zwiększenie liczby rozkazów ponad 16, gdyż jego działanie zależy od argumentu zawartego w rejestrze Oreg stanowiącego w tym przypadku rozszerzenie kodu operacji. W transputerze stosowane są rozkazy *opr* z argumentem 4- i 8-bitowym. Rozkaz *opr* z argumentem 4-bitowym jest kodowany w jednym bajcie (argument w polu *Data* jest automatycznie przenoszony do Oreg), natomiast rozkaz *opr* z argumentem 8-bitowym musi być poprzedzony rozkazem *pfix* przygotowującym w Oreg 4 b.zn. bity argumentu.

W transputerze stosowane są 3 sposoby adresacji:

- natychmiastowa,
- względna,
- pośrednia rejestrowa.

Adresacja natychmiastowa dotyczy argumentu bezpośredniego zawartego w Oreg i miejscem przeznaczenia może być jedynie stos obliczeniowy. W przypadku argumentu mniejszego od 16 (dziesiętnie) stała może być umieszczona w jednobajtowym rozkazie. Adresacja względna jest stosowana jedynie w rozkazach skokowych. Przesunięcie, pobierane jak poprzednio z rejestru Oreg, jest traktowane jako liczba ze znakiem dodawana do zawartości Iptr. Adresacja pośrednia odbywa się za pośrednictwem rejestrów Wptr (adresacja lokalna) lub Areg (adresacja nie-lokalna) przechowujących adres bazowy, do którego jest ewentualnie dodawane przesuniecie z Oreg. Przesuniecie jest liczone w słowach 32-bitowych.

Przy komunikowaniu się procesów istotną rolę pełnią rozkazy *in* i *out*. Pozwalają one na wymianę komunikatów pomiędzy dwoma procesami poprzez tzw. kanał, czyli słowo w pamięci przydzielone tym procesom. Po utworzeniu kanał jest oznaczony jako "pusty". Przed wydaniem rozkazu *in* lub *out* na stosie muszą być przygotowane dane dotyczące komunikatu: jego długość, adres początkowy i wskaźnik kanału. Pierwszy zgłaszający się proces znajduje kanał pusty, co oznacza, że nie może przeprowadzić transmisji. Następuje zapisanie stosu w obszarze roboczym a wskaźnik tego obszaru zostaje umieszczony w kanale i proces zawiesza się. Jeżeli

teraz do kanału zgłosi się drugi proces (*in* lub *out)* znajduje tam wskaźnik zawieszonego procesu i może przeprowadzić transmisję komunikatu; kanał zostaje ponownie oznaczony jako pusty a proces jest umieszczany na końcu kolejki procesów aktywnych. Analogicznie odbywa się komunikacja przez tzw. kanał zewnętrzny - gdy procesy są w rożnych transputerach i w transmisji muszą być zaangażowane łącza zewnętrzne DMA.

Istotną cechą transputera jest sprzętowa realizacja pewnych funkcji systemu operacyjnego związanych z szeregowaniem i obsługą procesów. Proces jest zdefiniowany przez swój obszar roboczy, w którym m.in. jest umieszczony adres następnego rozkazu i adres obszaru roboczego procesu następnego na liście procesów aktywnych. Za pomocą rozkazów sterujących procesami można inicjować wskazany proces czyli umieścić go na liście procesów aktywnych (*startp, runp*) i kończyć proces (*endp, stopp*). Ponadto grupa rozkazów *alt, altw, altend* pozwala na realizację konstrukcji ALT języka Occam służącej do wybrania z listy tego procesu, dla którego zostanie spełniony określony warunek (np. upływ czasu, dokonanie transmisji itp.). Rozkazy *talt, taltwt, enbt, dist* obsługujące obydwa zegary dostępne w procesorze umożliwiają sterowanie procesami uwarunkowanymi czasowo. Oprócz tego są rozkazy spełniające elementarne funkcje inicjacji i pamiętania stanu zegarów a także opóźnienia programu.

Dzięki temu, że architektura transputera została zaprojektowana dla wykonywania instrukcji języka Occam, kompilator tego języka daje bardzo wydajny i zwięzły kod binarny programu.

**Java Chips**

Język Java – zwany językiem Internetu, opracowany i propagowany przez firmę Sun, pozwala na pisanie programów działających na różnym sprzęcie. Cecha ta wynika z dwuetapowego tłumaczenia: najpierw program jest kompilowany do postaci pośredniej - tzw. *byte code* lub B-kodu (wg propozycji J.Bieleckiego [3]) - wymaganej przez hipotetyczną maszynę wirtualną Javy (JVM), a następnie B-kod jest interpretowany w języku wewnętrznym określonego procesora. Obecnie emulatory JVM są wbudowane w większość systemów operacyjnych i przeglądarek sieciowych, dzięki czemu tzw. aplety, czyli programy w Javie używane przez klienta, mogą być w

miarę potrzeby przesyłane w sieci w postaci zwięzłego B-kodu i bezpiecznie uruchamiane na miejscu.

Wykonywanie B-kodu odbywa się przez program interpretera lub metodą kompilacji-w-locie, czyli w czasie wykonywania (JIT, *just in time*). Obie metody mają swoje wady: interpretacja jest stosunkowo powolna natomiast kompilacja JIT, choć szybsza, wymaga większej pamięci co jest poważnym ograniczeniem dla procesorów wbudowanych. Rozwiązaniem tego dylematu może być sprzętowa realizacja JVM.

W 1996 r. został przedstawiony przez Sun Microsystems, Inc. procesor picoJava realizujący maszynę wirtualną Javy. W następnym roku sformułowano projekt serii nazwanej JavaChips i zawierającej trzy produkty: picoJava, microJava i UltraJava. Są to procesory o architekturze bazującej na architekturze picoJava i kolejno rosnącej wydajności, przewidziane do zastosowań w komputerach sieciowych, telefonach, grach video, sieciowych przystawkach do telewizorów (*set-top box*) i innych sterownikach sprzętu.

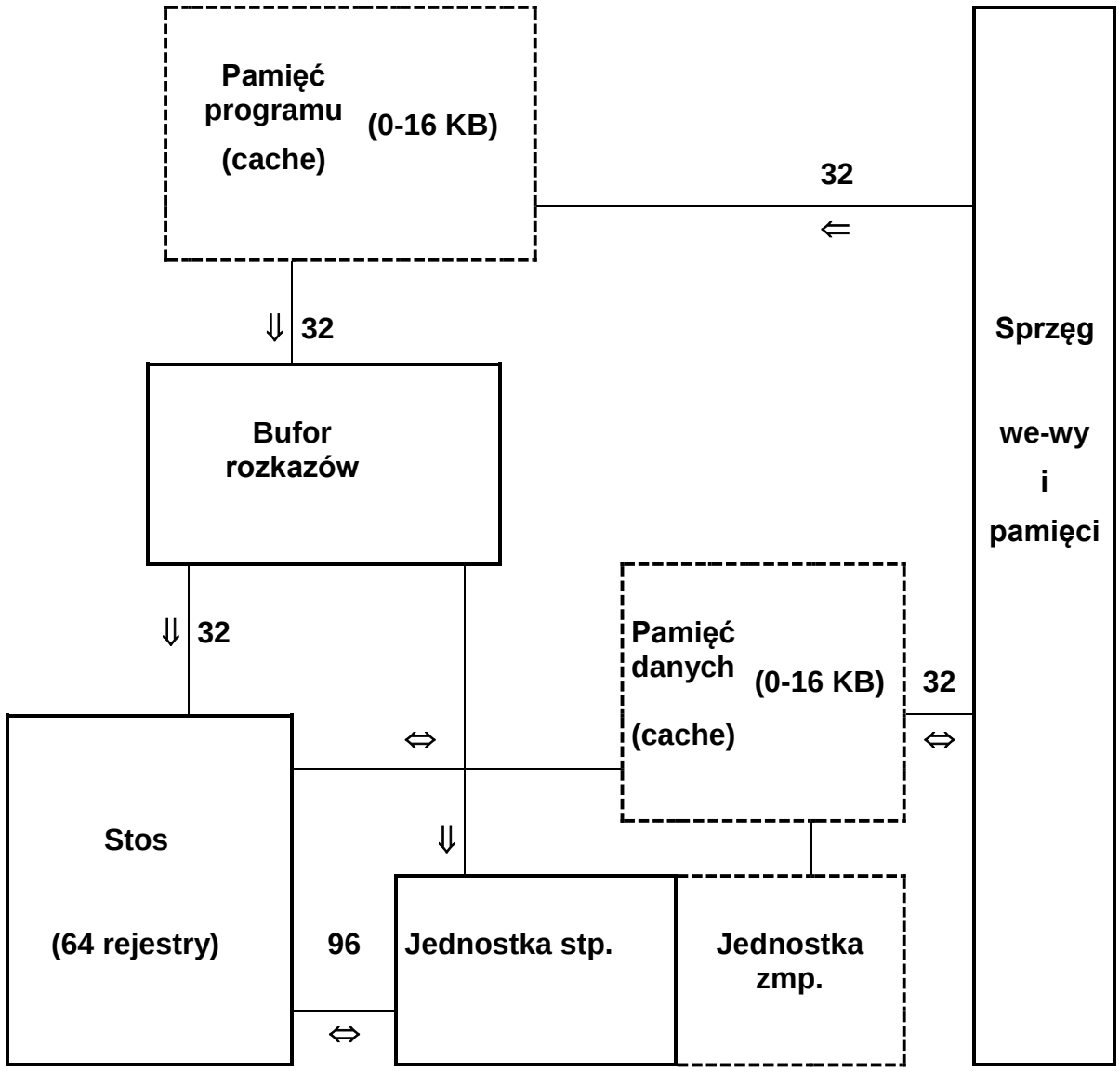
Architektura picoJava [15] łączy pewne elementy RISC i procesorów o organizacji stosowej. Nie wszystkie rozkazy maszyny wirtualnej są interpretowane układowo, niektóre – mające mniejszy wpływ na szybkość interpretacji B-kodu - są wykonywane mikroprogramowo, a kilka jest nawet emulowanych przez program. Rozkazy mają 1-bajtowy kod operacyjny i 0 – 4 bajtów argumentów; średnia długość rozkazu wynosi 1,8 bajta. Dzięki 4-stopniowemu potokowi (*pipeline*) zasilanemu z 12-bajtowego bufora rozkazów, większość rozkazów wykonuje się w czasie 1-3 cykli.

Inaczej niż w klasycznej architekturze RISC rozwiązano sprawę rejestrów – w picoJavie ich rolę spełnia stos układowy zrealizowany jako cykliczny bufor 64-pozycyjny; w przypadku przepełnienia lub wyczerpania stosu jego kontynuacją staje się automatycznie pamięć danych.

Założenia projektowe przewidywały możliwość doboru konfiguracji procesora do różnych zastosowań w systemach wbudowanych, stąd niektóre elementy picoJava są opcjonalne pozwalając projektantowi systemu na wybór mniej lub bardziej kosztownej wersji układu. I tak obie pamięci cache – zarówno pamięć danych jak i programu - mogą mieć wielkość od 0 do 16kB, podobnie - można włączyć lub wyłączyć ze struktury jednostkę zmiennoprzecinkową. W przypadku braku jednostki

zmiennoprzecinkowej odpowiednie rozkazy są wykonywane przez emulację programową.

Porównanie wydajności 3 procesorów: i486, Pentium i symulatora picoJava, działających z tą samą częstotliwością zegara, wykazało znaczną przewagę picoJavy zarówno w stosunku do interpretacji jak i kompilacji JIT. Badanie przeprowadzono metodą programów testowych (*benchmarks*) o dużej złożoności, jednym z nich był np. obiektowy kompilator Javac liczący ok. 25 tys. linii kodu Javy. picoJava okazała się 14-krotnie szybsza niż Pentium pracujące z interpreterem a 5-krotnie szybsza niż działający na Pentium kompilator JIT.




**Struktura procesora picoJava.**

## Zakończenie

Z przedstawionych tutaj realizacji niewątpliwie szczególne znaczenie ma język Java i związane z nim mikroprocesory. Wynika to zarówno z zalet samego języka jak i z pozycji rynkowej głównego sponsora i jego możnych sprzymierzeńców (m.in. Toshiba, Mitsubishi, Samsung, a ostatnio (marzec 1998) – IBM). Entuzjaści (por. [2]) uważają nawet że Java stanowi zwieńczenie dotychczasowego rozwoju języków programowania dając wreszcie odpowiednie narzędzie profesjonalnym programistom. Tempo rozpowszechniania się języka jest imponujące, co ilustrują dane z początku tego roku (wg Sun Microsystems [6]):

- od pojawienia się Javy w 1995 r. opublikowano ponad 800 książek na temat programowania w tym języku,
- ponad 200 uczelni w USA prowadzi zajęcia z Javy,
- ponad 150 firm kupiło licencję Java,
- z serwera Sun ściągnięto ponad 1 mln kopii pakietu programistycznego JDK 1.1 zawierającego kompilator Javy i programy narzędziowe,
- jest ok. 70 mln stanowisk komputerowych wyposażonych w Javę.

Rozwijana od kilku lat w skali międzynarodowej i międzypaństwowej koncepcja Globalnej Infrastruktury Informacyjnej (GII) implikuje proliferację urządzeń, których jądrem jest odpowiednio oprogramowany mikroprocesor. Jednym z aspektów GII jest *nomadicity*, czyli zdolność systemu do komunikowania się z obiektem mobilnym – w sytuacji, gdy przemieszczenie następuje w skali globalnej. Wydaje się, że procesory interpretujące kod Javy są predestynowane do budowy takich obiektów. Zaczątek systemów mobilnych stanowi obecnie telefonia komórkowa czy satelitarny system pozycjonowania obiektów (GPS). Obiecującym celem zastosowań JavaChips są tanie *set-top box* ("Internet dla ubogich") jak również komputery sieciowe (NC), będące w zamierzeniu alternatywą dla pecetów.

Także w zastosowaniach nie-sieciowych – w elektronice użytkowej i przemyśle rozrywkowym masowo występują gadżety sterowane mikroprocesorami. Perspektywy wydają się nieograniczone; w klasyfikacji oprogramowania według środowiska użytkownika można już spotkać nie tylko tradycyjne zastosowania w grach video (*gameware*) lub bankach (*bankware*) ale też "biurkowe" (*deskware*), czy zgoła "kuchenne" (*kitchenware)* a nawet "łazienkowe" (*bathroomware)* [5]. Od

Pathfindera na Marsie do tamagochi popiskujących w przedszkolu, wszędzie jądro systemu stanowią mikroprocesory, które zwykle wymagają zwięzłego kodu przy stosunkowo prostym interfejsie i bardzo wyspecjalizowanych funkcjach. Dla wielu z tych urządzeń właściwym rozwiązaniem może być sprzętowy interpreter języka typu Forth czy obecnie – Java.

**Literatura**


1. Bell C.G., Newell A.:Computer Structures: Readings and Examples. McGraw-Hill Book Co. 1971.
2. Bielecki J.: Droga do Javy. *Informatyka*, 1997, nr 9.
3. Bielecki J.: Java – słownik terminów. *Informatyka*, 1997, nr 1.
4. Bilski E.: Okres maszyn cyfrowych typu ODRA. *Informatyka*, 1989, nr 8-12.
5. Bushnell N.: Relationships between Fun and the Computer Business. *Comm of the ACM*, August 1996. Vol.39, nr 8.
6. *Computerworld* 1998, nr 9
7. Grier D.A.: The ENIAC, the Verb “to program” and the Emergence of Digital Computers. *IEEE Annals of the History of Computing*, Vol.18, No1, Spring 1996.
8. Komorowski W.: Architektura mikroprocesorów. *Informatyka* 1997, nr 12.
9. Koopman P.J.: Stack Computers: the new wave. Ellis Horwood, 1989. (książka dostępna również on-line w Internecie)
10. Lonergan W., King P.: Design of the B 5000 System. *Datamation* vol. 7, no. 5. May 1961. (w [1])
11. Matthews J: Forth. Applications in Engineering and Industry. Ellis Horwood Ltd., 1989.
12. Mieścicki J.: Uniwersalna maszyna cyfrowa UMC-1. [w “Laboratorium organizacji maszyn cyfrowych” pod red. K.Fiałkowskiego, Wyd. Pol.W., Warszawa 1968]
13. Mikanik W.: Charakterystyka języka OCCAM. *Informatyka*, 1996, nr 11.
14. Money S.A.: Mikroprocesory. WKiŁ. Warszawa 1996.
15. O’Connor J.M., Tremblay M.: picoJava-I: The Java Virtual Machine in Hardware. *IEEE Micro*, March/April 1997.
16. Organick E.I.: A Programmer’s View of the Intel 432 System. McGraw-Hill Book Co., 1983.
17. Whitworth I.R.: 16-bit Microprocessors. Collins. London, 1987.
18. A Guide to RISC Microprocessors (ed. Slater M.). Academic Press, Inc., 1992.


*Tekst powyższy zawiera treść artykułu zamieszczonego w miesięczniku “Informatyka” nr 7/8 z 1998 r.*