1 1989

informatyka

Pro domo sua
Pakiet syntezy mowy
Metaasembler mikroprogramów

WYDAWNICTWO NOT SIGMA

Cena 300,– zł

informatyka

1 1989

Nr 1

Miesięcznik Rok XXIV
Styczeń 1989

Organ Komitetu
Naukowo-Technicznego NOT
ds. Informatyki

Materiałów nie zamówionych redakcja nie zwraca

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

RSW „PRASA-KSIĄŻKA-RUCH"
PRASOWE ZAKŁADY GRAFICZNE
ul. Dworcowa 13, 85-950 BYDGOSZCZ
Zam. 4226/88.
Obj. 4,0 ark. druk. Nakład 7950 egz.

ISSN 0542-9951. INDEKS 36124

Cena egzemplarza 300 zł
Prenumerata roczna 3600 zł

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

W NUMERZE

W NAJBLIŻSZYCH NUMERACH:

- Władysław M. Turski „Co należy i można zrobić"
- Andrzej Blikle „O uwarunkowaniach rozwoju systemów informatycznych i przemysłu oprogramowania"
- Janusz Zalewski „Nie sama mikroelektronika"
- Wnioski V Krajowej Konferencji Informatyków
- Uchwała II Walnego Zjazdu Delegatów Polskiego Towarzystwa Informatycznego
- Osobiste refleksje z V Krajowej Konferencji Informatyków
- Bruno Lamborghini „Wpływ informacji i technologii informacyjnej na strukturę firmy"
- Zygmunt Bieńko „Kryzys polskiej informatyki"

Pro domo sua

Dobiegł końca kolejny rok pracy zespołu redakcyjnego INFORMATYKI. Mamy więc okazję dokonać podsumowania naszej dotychczasowej działalności i wznowić dyskusję nad profilem pisma.

Jest rzeczą oczywistą, że INFORMATYKA, będąc – jak dotychczas – jedynym, w pełni profesjonalnym czasopismem informatycznym w Polsce, musi sprostać wielu, często sprzecznym ze sobą wymaganiom. Musi więc zaspokajać potrzeby Czytelników o zróżnicowanym poziomie zawodowym, mieć dość szeroki zakres tematyczny, informować o tendencjach rozwojowych światowej informatyki, omawiać konkretne rozwiązania techniczne (sprzętowe i programistyczne), a jednocześnie nie stronić od tematyki naukowej. Aby sprostać – choćby częściowo – tym oczekiwaniom, należałoby dokonać oceny kondycji informatyki i ściśle uzależnić profil pisma od bieżących potrzeb potencjalnych czytelników.

Wprawdzie sprawie polskiej informatyki zamierzamy poświęcić oddzielny numer, ale nie popełnię chyba błędu, jeśli stwierdzę już teraz, że jej obraz rysuje się niepokojąco na tle światowego rozwoju, a poziom szeroko rozumianej komputeryzacji kraju nie napawa szczególnym optymizmem. Jakie wynikają stąd wnioski dla redakcji? Przy założeniu, że bariera technologiczna i sprzętowa jest w Polsce – w dającej się przewidzieć przyszłości – nie do pokonania, powinniśmy skoncentrować się głównie na upowszechnianiu metod inżynierii oprogramowania oraz nowych technik programowania.

Wbrew powszechnie panującemu przekonaniu słabością polskiej informatyki jest bowiem nie tylko brak nowoczesnego sprzętu komputerowego, lecz również – a w moim przekonaniu przede wszystkim – dość niski poziom kultury programistycznej. Dostępne w kraju komputery, i to zarówno komputery starszej generacji i nowocześniejsze mikrokomputery, umożliwiają – pomimo ich ograniczeń – tworzenie ciekawego i nowatorskiego oprogramowania. Dowodzą tego niektóre, niestety nieliczne, krajowe zastosowania. Pierwszoplanowym zadaniem INFORMATYKI jest więc prezentacja rozwiązań programistycznych, ukazujących możliwości wykorzystania komputerów w różnych obszarach działalności ludzkiej oraz popularyzacja nowych metod rozwiązywania problemów i technik programowania stosowanych w poszczególnych działach informatyki, zwłaszcza w najaktywniej rozwijanych obecnie działach, takich jak bazy danych, przetwarzanie rozproszone, sztuczna inteligencja lub grafika komputerowa.

Ukierunkowanie INFORMATYKI na zagadnienia programistyczne ma obecnie szczególne znaczenie. Otóż wraz ze wzrostem liczby mikrokomputerów obserwuje się zjawisko swego rodzaju analfabetyzmu informatycznego. Mikrokomputer stał się narzędziem pracy nie tylko dla wielu doświadczonych informatyków, lecz także dla tych, którzy nie mieli dotychczas żadnej styczności z informatyką. Uważają się oni, a co gorsza są uważani nierzadko, za informatyków. Stan ten jest oczywiście spowodowany brakiem odpowiednio licznej i wykwalifikowanej kadry informatycznej. Ustalając profil naszego pisma, będziemy starali się uwzględnić również potrzeby tych Czytelników, którzy od niedawna są związani zawodowo z komputerami i którzy potrzebują dodatkowego wsparcia informatycznego i uzupełniania wiedzy.

Jakość tworzonego oprogramowania jest nie tylko pochodną wiedzy informatycznej programisty, lecz także pochodną motywacji finansowej i konkurencji środowiskowej. I w tym zakresie INFORMATYKA może odegrać pewną rolę, opowiadając się aktywnie za objęciem zasadami prawa autorskiego również działalności programistycznej.

Wysuwając na pierwszy plan zagadnienia budowy oprogramowania, nie zamierzamy zapominać o sprzęcie komputerowym. Wręcz przeciwnie, im bardziej oddalają się od nas uprzemysłowione kraje Zachodu, tym szerzej musimy informować o komputerowych nowościach. Na naszych łamach znajdziemy miejsce dla nowych architektur komputerowych, systemów wieloprocesorowych, superkomputerów, specjalizowanych maszyn lispowych i prologowych – nawet jeśli nie pojawią się one zbyt prędko w naszym kraju. Szczególną uwagę chcemy poświęcić stacjom roboczym, czyli komputerowym stanowiskom pracy o dużej mocy obliczeniowej i zintegrowanym środowisku programowania. Pierwowzorami stacji roboczych są masowo produkowane mikrokomputery Macintosh, IBM PC i IBM PS/2. W profesjonalnych zastosowaniach komputery te są wybierane obecnie przez stacje robocze Sun, Apollo, MicroVAX i IBM PC/RT, a ostatnio przez superstacje robocze Ardent; one też wcześniej czy później pojawią się na naszym rynku.

Jak wygląda w tym kontekście przekrój tematyczny artykułów opublikowanych na łamach INFORMATYKI w ciągu trzech lat działalności obecnego zespołu redakcyjnego? W wykazie przytoczonym w spisie treści poprzedniego rocznika widoczna jest supremacja artykułów poświęconych językom programowania, systemom operacyjnym i sprzętowi, co można wytłumaczyć dużym zainteresowaniem mikrokomputerami IBM PC. W tym kontekście całkiem nieźle prezentują się komunikacja i sieci komputerowe oraz bazy danych, nieco gorzej – inżynieria oprogramowania, sztuczna inteligencja, podstawowe narzędzia i pakiety oraz architektury komputerowe. Zdecydowanie zbyt mało jest artykułów poświęconych grafice komputerowej, algorytmom obliczeniowym i dydaktyce. Na podkreślenie zasługuje prawie całkowity brak opisów zastosowań (np. w tak ważnych obecnie dziedzinach jak automatyzacja produkcji, automatyzacja projektowania i automatyzacja prac biurowych). Brakuje również ogólnych artykułów dotyczących programów i prognoz rozwoju informatyki, a także roli komputerów w społeczeństwie. Podsumowując można stwierdzić, że treść opublikowanych artykułów odpowiada w znacznej mierze postulowanej tematyce, lecz niezbędne wydaje się nieco inne rozłożenie akcentów na poszczególne tematy. W szczególności należy zmniejszyć liczbę artykułów o językach programowania i systemach operacyjnych, a zwiększyć liczbę artykułów poświęconych zastosowaniom.

INFORMATYKA ma do odegrania w środowisku informatycznym niezwykle istotną rolę: rolę jedynego pisma zawodowego, pisma uczącego, pisma informującego, wreszcie pisma zastępującego wielu Czytelnikom (na ogół niedostępną w kraju) literaturę obcojęzyczną. O tematyce i poziomie publikowanych materiałów decyduje w dużym stopniu zespół redakcyjny. Przeważająca część artykułów pojawia się bowiem na łamach pisma z inicjatywy członków redakcji. Brakuje nam jednak aktywniejszego kontaktu z Czytelnikami. Chodzi zarówno o nadsyłanie artykułów, którymi Autorzy zechcieliby się podzielić z Czytelnikami, jak i o polemiki, wypowiedzi i opinie Czytelników. Poziom naszego czasopisma jest odbiciem poziomu krajowego środowiska informatycznego. Można powiedzieć, że INFORMATYKĘ wydajemy wspólnie i wspólnie za nią odpowiadamy.

Na marginesie naszej dyskusji odnotowujemy pozytywny fakt zmiany drukarni, w której co miesiąc rodzi się INFORMATYKA. Liczymy na to, że pozbędziemy się tym samym części naszych bolączek, które tak utrudniały życie Czytelnikom i redakcji. Zmiana ta pozwala skrócić cykl wydawniczy i zapewni – miejmy nadzieję – terminowość ukazywania się czasopisma. Jakość druku uległa poprawie i pojawiły się wreszcie brakujące dotychczas znaki specjalne. Zmianie ulega również szata graficzna pisma. Potraktujemy te pierwsze od wielu lat dobre wiadomości jako zwiastun dalszych zmian, które pozwolą nam wydawać pismo na miarę naszych wspólnych oczekiwań.

MAREK MACHURA

LESLIE LAMPORT
Digital Equipment Corporation
Systems Research Center
Palo Alto, stan Kalifornia
USA

Uproszczone specyfikowanie systemów współbieżnych (3)

W ostatniej części artykułu omówiono specyfikowanie właściwości żywotności oraz inne zagadnienia dotyczące systemów współbieżnych.

WŁAŚCIWOŚCI ŻYWOTNOŚCI

Właściwości żywotności zakładają, że coś musi się zdarzyć. W specyfikacji metodą aksjomatu przejść zdarzeniami są zmiany wartości funkcji stanu. To, co musi się zdarzyć, jest wyrażone przez jawne aksjomaty określające, jak te wartości ostatecznie muszą się zmieniać.

Aksjomaty specyfikujące żywotność są zapisywane w języku logiki temporalnej, będącej rozszerzeniem zwykłej logiki o dwa operatory temporalne, oznaczone kwadratem $\Box$ (czytanym *„odtąd"*, ang. *henceforth*) i rombem $\Diamond$ (czytanym *„ostatecznie"*, ang. *eventually*). Formuła $\Box P$ zakłada, że P jest prawdziwe teraz i w każdym czasie w przyszłości, a formuła $\Diamond P$ zakłada, że P jest prawdziwe teraz i w niektórych chwilach w przyszłości. Ponieważ P jest ostatecznie prawdziwe wtedy i tylko wtedy, gdy jest nie zawsze fałszywe, to $\Diamond P$ jest równoważne formule $\neg \Box \neg P$. Tę równoważność omówiono dokładniej w [3]. Wygodne jest również zdefiniowanie operatora $\rightarrow$ (czyt. *„prowadzi do"*, ang. *leads to*). Służy on do zastąpienia formuły $\Box(P \Rightarrow \Diamond Q)$ przez formułę $P \rightarrow Q$ i zakłada, że jeśli P stanie się prawdziwe, to Q będzie prawdziwe wtedy lub w pewnej chwili w przyszłości[1]. Bardziej szczegółowy wykład logiki temporalnej przedstawiono w [6].

W specyfikacji automatu z wodą sodową (cz. 1 artykułu) można postawić wymaganie, że jeśli użytkownik wrzucił dostateczną liczbę monet, to automat musi mu ostatecznie wydać puszkę. Jest to wyrażone formułą $(f = III) \rightarrow (f = I)$, która zakłada, że jeśli $f = III$, to f musi być ostatecznie równe I.

Specyfikacja automatu z wodą sodową prawdopodobnie nie będzie zawierać innych aksjomatów żywotności, ponieważ nie zakłada się, że użytkownik musi wrzucić monetę. Jednakże, można postawić wymaganie, że jeśli wrzuci on jedną monetę ćwierćdolarową, to musi wrzucić również następną, co jest stwierdzone przez aksjomat $(f = II) \rightarrow (f = III)$.

W specyfikacji automatu z wodą sodową przedstawionej na wydruku 1 w poprzedniej części artykułu, zakłada się że następna akcja musi być ostatecznie wykonana, chyba że jest to akcja γ, której użytkownik nie musi wykonywać w ogóle. Jeżeli następną akcją jest akcja α, to można to stwierdzić w asercji $(pc = \alpha) \rightarrow (pc = \beta)$. Jednakże, można to również ująć słabszą asercją $(pc = \alpha) \rightarrow (pc \neq \alpha)$, ponieważ z wydruku wynika, że jeśli $pc = \alpha$, to jedyną możliwą wartością, którą może przybrać pc w wypadku zmiany, jest wartość β. Pełna specyfikacja żywotności dla tego przykładu jest następująca:

$$\forall\, \xi \neq \gamma : (pc = \xi) \rightarrow (pc \neq \xi) \qquad (1)$$

Aksjomaty żywotności są oczywiste, gdy spojrzy się na rysunek z pierwszej części i wydruk 1 z drugiej części artykułu. Czy jednak nie można niejawnie wbudować aksjomatów żywotności do języka zamiast zapisywać je oddzielnie?

Niektóre aksjomaty żywotności można wbudować niejawnie do języka. Jednakże, warunki żywotności występujące w specyfikacjach są zbyt różnorodne, aby można je wyrazić niejawnie za pomocą rozsądnego zbioru konstrukcji językowych.

Nieformalne wymaganie żywotności dla specyfikacji bazy danych z wydruku 4 w poprzedniej części artykułu jest takie, aby każda operacja zachowana w multizbiorze saved_ops była ostatecznie wykonana. Jak to wyrazić formalnie?

Pierwszą próbą wyspecyfikowania tej właściwości może być aksjomat
$\forall\, (o, r, v): (o, r, v) \in saved\_ops \rightarrow (o, r, v) \notin saved\_ops$, który zakłada, że jeśli trójka (o, r, v) należy do multizbioru *sawed_ops*, to ostatecznie przestanie do niego należeć. Z pozostałej części specyfikacji powinno wynikać, że jedynym sposobem usunięcia tej trójki z multizbioru jest wykonanie odpowiedniej operacji w bazie danych zgodnie z akcją γ.

Ten aksjomat wyrażałby żądane wymaganie, gdyby multizbiór *saved_ops* nie mógł zawierać dwóch egzemplarzy jednej trójki. Jednakże, gdyby ta sama trójka (o, r, v) była ciągle umieszczana w multizbiorze przez różne wywołania procedury *exec*, to multizbiór *saved_ops* mógłby zawsze zawierać egzemplarz (o, r, v), wskutek czego aksjomat nie byłby spełniony. Wszystkim, co można założyć jest to, że jeśli pewna trójka (o, r, v) należy do multizbioru *saved_ops*, to ostatecznie przynajmniej jeden jej egzemplarz będzie usunięty, tzn. istnieje akcja γ, która ostatecznie usuwa egzemplarz (o, r, v)[2]. Formuły dotyczą jednak stanów, nie akcji. Asercja o wystąpieniu akcji γ ma postać formuły temporalnej zakładającej, że w pewnej chwili multizbiór zawiera k egzemplarzy trójki, a w jakiś czas później zawiera ich mniej. Pożądany warunek żywotności wyraża się następującym aksjomatem, w którym $e \# B$ oznacza liczbę egzemplarzy elementu e w multizbiorze B:

$$\forall\, (o, r, v): [(o, r, v) \in saved\_ops\ ] \rightarrow$$
$$[\exists\, k \text{ takie że } ((o, r, v) \# saved\_ops = k)$$
$$\wedge \Diamond ((o, r, v) \# saved\_ops < k)]$$

Można wprowadzić notację, która ułatwia utworzenie asercji, że określona akcja ostatecznie wystąpi, co umożliwia zapisanie aksjomatu w następującej postaci:

$$[(o, r, v) \in saved\_ops\ ] \rightarrow \gamma\ (o, r, v)$$

Jednakże, wyjaśnienie takiej notacji prowadziłoby do zagadnień projektowania języka, czego nie chciałbym tutaj omawiać.

Czy operatory oznaczone kwadratem i rombem, a także trzeci operator zdefiniowany przy ich użyciu, są wszystkim, czego potrzeba do specyfikowania właściwości żywotności?

Tak.

---

[1] W oryginale Autor używa wężyka zakończonego strzałką. Ponieważ w tej części atrykułu zwykła strzałka nie występuje w innym znaczeniu, użyto jej dla oznaczenia *operator* w formule $P \rightarrow Q$ (przyp. red.).

[2] *Trzeba zauważyć, że identyczne trójki są nieodróżnialne, tak więc nie ma sensu pytanie, który egzemplarz zostaje usunięty.*

Jak można zweryfikować, że realizacja spełnia właściwości żywotności specyfikacji?

Należy zweryfikować każdy aksjomat żywotności. Rozważmy następujący aksjomat żywotności dla specyfikacji z rysunku w pierwszej części artykułu:

$$(f = III) \rightarrow (f = I) \qquad (2)$$

Aby udowodnić, że wydruk 1 z drugiej części artykułu jest realizacją tej specyfikacji, zdefiniowano f w zależności od funkcji stanu realizacji, x, y i pc, co przedstawiono na wydruku 2. Podstawiając to wyrażenie na f do powyższej formuły, otrzymuje się:

$$[(pc = \beta \wedge x = 50) \vee (pc = \delta \wedge x + y = 50) \vee (pc = \varepsilon)] \rightarrow [(pc = \alpha) \vee ...] \qquad (3)$$

Aby zweryfikować, że realizacja spełnia aksjomat (2), należy udowodnić, że z aksjomatów żywotności i właściwości bezpieczeństwa specyfikacji z wydruku 1 wynika formuła (3). Takie postępowanie ma sens, ponieważ ta formuła jest wyrażeniem wiążącym funkcje stanu realizacji.

Na podstawie aksjomatu żywotności (1) i właściwości bezpieczeństwa można ustalić następujący łańcuch relacji $\rightarrow$, przyjmując że formuła $A \rightarrow B \rightarrow C$ jest skróconym zapisem formuły $(A \rightarrow B) \wedge (B \rightarrow C)$:

$$(pc = \delta \wedge x + y = 50) \rightarrow (pc = \beta \wedge x = 50) \rightarrow (pc = \varepsilon) \rightarrow (pc = \alpha)$$

Przykładowo, aby zweryfikować, że

$$(pc = \beta \wedge x = 50) \rightarrow (pc = \varepsilon)$$

trzeba zauważyć, iż z formuły (1) wynika, że ostatecznie $pc \neq \beta$, a z aksjomatów przejść wynika, że jeśli $pc = \beta$ i $x = 50$, to wartość pc może zmienić się tylko na ε. Jak widać, w dowodzie korzysta się z właściwości bezpieczeństwa i żywotności realizacji.

Czytelnikom pozostawia się do sprawdzenia, czy powyższy łańcuch relacji $\rightarrow$ intuicyjnie prowadzi do formuły (3). Formalną metodę stanowiącą podstawę tego nieformalnego rozumowania przedstawiono w [6].

Jaką ogólną metodę zastosowano w tym przykładzie?

Przypomnijmy, że specyfikacja formalna jest formułą:

$$\exists\, f_1, ..., f_n \text{ takie, że } X$$

gdzie f_i są wewnętrznymi funkcjami stanu, a X jest formułą określającą, jak zmieniają się wartości wewnętrznych funkcji stanu i funkcji stanu sprzężenia. Podobnie, realizacja jest reprezentowana formułą:

$$\exists\, h_1, ..., h_m \text{ takie, że } Y$$

gdzie h_i są wewnętrznymi funkcjami stanu realizacji, a Y jest formułą wiążącą funkcje h_i z funkcjami stanu sprzężenia. Poprawność realizacji jest wyrażona formułą:

$$(\exists\, h_1, ..., h_m \text{ takie, że } Y) \supset (\exists\, f_1, ..., f_n \text{ takie, że } X)$$

Tej formuły dowodzi się wyrażając funkcje stanu specyfikacji f_i w zależności od funkcji stanu realizacji h_i i dowodząc, że $Y \supset \bar{X}$ gdzie $\bar{X}$ jest formułą otrzymaną z X przez podstawienie zamiast funkcji f_i odpowiednich wyrażeń zawierających funkcje h_i.

Podział specyfikacji na wymagania bezpieczeństwa i żywotności można wyrazić zapisując:

$$X = X_s \wedge X_l$$

gdzie X_s oznacza aksjomaty bezpieczeństwa, a X_l – aksjomaty żywotności oraz podobnie:

$$Y = Y_s \wedge Y_l$$

Dowodząc, że spełnione są właściwości bezpieczeństwa specyfikacji, co czyni się wykazując, że każda akcja realizacji zmieniająca funkcje stanu specyfikacji wykonuje to, co jest dozwolone przez pewną akcję specyfikacji, dowodzi się, że:

$$Y_s \supset X_s$$

Aby udowodnić, że spełnione są właściwości żywotności specyfikacji, dowodzi się, że:

$$(Y_s \wedge Y_l) \supset X_l$$

Inaczej mówiąc, w celu udowodnienia właściwości żywotności specyfikacji używa się zarówno właściwości bezpieczeństwa jak i właściwości żywotności realizacji.

INNE ZAGADNIENIA

Na zakończenie omówię jeszcze wenętrzne funkcje stanu, współbieżność oraz modularność i dekomponowanie specyfikacji.

Wewnętrzne funkcje stanu

Choć specyfikacja powinna dotyczyć tylko zewnętrznie obserwowalnego zachowania systemu, w metodzie aksjomatu przejść wprowadza się wewnętrzne funkcje stanu i przejścia wewnętrzne. Czy nie stanowi to nadmiernego ograniczenia specyfikacji?

Aby wyspecyfikować zewnętrzne obserwowalne zachowanie, należy opisać wszystkie dozwolone ciągi akcji sprzężenia. W większości konwencjonalnych metod specyfikowania ciągu akcji używa się niejawnych stanów wewnętrznych. Przykładowo, gramatyka bezkontekstowa jest równoważna automatowi, którego stany są niejawne w tej gramatyce. Rachunek CCS [4] można uważać za pojedynczy automat, którego stany są zbiorem formuł tego rachunku. Gramatyki bezkontekstowej lub rachunku CCS można łatwo użyć jako języka do wyrażania aksjomatów przejść.

Logiki temporalnej i innych metod aksjomatycznych używa się do pisania specyfikacji, które nie dotyczą stanów wewnętrznych. Czy te specyfikacje nie są ogólniejsze niż specyfikacje w metodzie aksjomatu przejść?

Niech specyfikacja, która nie używa stanów wewnętrznych, nazywa się specyfikacją czysto temporalną. W raporcie [1] wykazano, że specyfikacje czysto temporalne nie są ogólniejsze niż specyfikacje w metodzie aksjomatu przejść. Autorzy raportu zdefiniowali logikę, co najmniej tak efektywną jak większość logik używanych do specyfikacji czysto temporalnych, i wykazali, że każda formuła ich logiki jest równoważna asercji o automacie opartym na tej formule. Taki automat można zinterpretować jako specyfikację metodą aksjomatu przejść, która jest równoważna specyfikacji czysto temporalnej, reprezentowanej przez formułę oryginalną.

Jeśli nawet specyfikacje czysto temporalne nie są logicznie ogólniejsze niż specyfikacje metodą aksjomatu przejść, to czy uniknięcie w nich jawnego wyrażania wewnętrznych funkcji stanu nie oznacza w praktyce, że są one mniej podatne do nadmiernego ograniczania realizacji?

W metodzie aksjomatu przejść, znacznie łatwiej niż w metodach czysto temporalnych opisuje się poszczególne realizacje zamiast specyfikowania tylko pożądanego zachowania sprzężenia. Wyeliminowanie wewnętrznych funkcji stanu wymaga bowiem użycia złożonych formuł temporalnych. Czytelnicy mogą ocenić dodatkową złożoność niezbędną do wyspecyfikowania zachowania metodami czysto temporalnymi, pisząc językiem potocznym dwie nieformalne specyfikacje rejestru:

- pierwszą, używającą zawartości rejestru (tworzących wewnętrzną funkcję stanu),
- drugą, czysto temporalną, w której jest mowa tylko o operacjach odczytu i zapisu, bez uwzględnienia zawartości rejestru.

Jak stwierdziłem w praktyce, specyfikacje czysto temporalne są trudno zrozumiałe. Choć są one mniej podatne do „przespecyfikowania" systemu, to są o wiele bardziej podatne na „niedospecyfiko-

wanie" go przez pominięcie istotnych ograniczeń. Metody czysto temporalne są trudne do stosowania w praktyce, ponieważ nie określają, gdzie należy zacząć specyfikację (które właściwości powinny być wyspecyfikowane jawnie, a które powinny wynikać z innych właściwości) lub kiedy należy ją zakończyć (czy wszystkie pożądane właściwości wynikają ze specyfikacji). Metoda aksjomatu przejść natomiast zapewnia strukturalne podejście do pisania specyfikacji: najpierw wybiera się funkcje stanu, następnie specyfikuje, jak mogą się zmieniać (aksjomaty przejść), a na końcu specyfikuje, kiedy muszą się zmieniać (aksjomaty żywotności).

Dowodzenie poprawności realizacji wymaga zdefiniowania funkcji stanu specyfikacji w zależności od funkcji stanu realizacji. Czy nie ma wypadków, gdy jest to niemożliwe, ponieważ niektóre funkcje stanu specyfikacji są niepotrzebne i w rzeczywistości nie są realizowane?

To prawda. Jednym przykładem jest program, rozumiany jako specyfikacja swojej skompilowanej wersji, w której optymalizujący kompilator eliminuje zmienną lokalną, ustawianą, ale nigdy nie odczytywaną. Nie realizowane funkcje stanu nie muszą być jednak niepotrzebne. Można sobie wyobrazić specyfikację, która na początku określa podjęcie decyzji przez system, czy ma on działać jako automat z wodą sodową czy jako baza danych, by następnie działać według wybranej zasady. Ta absurdalna specyfikacja opisuje funkcje stanu zarówno dla automatu z wodą sodową, jak i dla bazy danych. Jednakże specyfikacja może być spełniona albo przez realizację automatu z wodą sodową, albo przez realizację bazy danych, bez realizowania funkcji stanu tej drugiej części.

Jak dowodzi się poprawności realizacji, jeżeli nie realizuje ona wyspecyfikowanych funkcji stanu?

Dowodząc poprawności realizacji, można do niej dodać tzw. pomocnicze funkcje stanu. Pomocnicza funkcja stanu jest podobna do zmiennej pomocniczej dodawanej przy dowodzeniu poprawności programu współbieżnego [5]. Jest wewnętrzną funkcją stanu, dodawaną w taki sposób, że nie zmienia specyfikacji, jak powinny się zmieniać „rzeczywiste" funkcje stanu. Istniejące aksjomaty przejść są modyfikowane tak, aby wskazać, jak zmieniają się pomocnicze funkcje stanu.

Czy dodając pomocnicze funkcje stanu nie dowodzi się poprawności nowej realizacji, z dodatkowymi funkcjami stanu, zamiast – realizacji pierwotnej?

Tak nie jest. Aby to zrozumieć, należy uświadomić sobie formalne znaczenie kwantyfikacji egzystencjalnej nad funkcjami stanu. Intuicyjnie, formuła

$\exists h$ takie, że A

zakłada istnienie h nie w „świecie rzeczywistym", gdzie jedynymi istniejącymi funkcjami stanu są opisane przez realizację, lecz w „świecie fikcyjnym", gdzie przyjmuje się istnienie każdej możliwej funkcji stanu. Pomocnicze funkcje stanu nie zmieniają realizacji, lecz służą do konstruktywnego dowodu istnienia pewnych dopuszczalnych funkcji stanu. Można przepisać dowód poprawności, eliminując pomocnicze funkcje stanu, lecz taki dowód byłby trudniejszy do zrozumienia.

Sytuacja, w której funkcji stanu specyfikacji nie można wyrazić w zależności od funkcji stanu realizacji, jest nietypowa. Podobnie jak dobry program nie oblicza wartości, które nigdy nie są używane, dobra specyfikacja nie zawiera funkcji stanu, które nie są potrzebne. Specyfikacje pozostawiające realizatorowi wybór, które funkcje stanu zrealizować, są rzadkie. W praktyce, nie specyfikuje się systemu, który może wybrać, czy będzie działał jako automat z wodą sodową, czy jako baza danych. Dlatego pomocnicze funkcje stanu są rzadko potrzebne. Osobiście radziłbym nie wprowadzać ich w celu łatwiejszego wyrażenia funkcji stanu specyfikacji, ponieważ wyrażając te funkcje stanu w języku funkcji stanu „rzeczywistej" implementacji, można nauczyć się bardzo wiele o samej implementacji.

Współbieżność

Metoda aksjomatu przejść służy do specyfikowania systemów współbieżnych, lecz zachowanie systemu opisuje się w niej jako ciąg akcji. Gdzie w takim razie tkwi współbieżność?

Podstawą prawie wszystkich metod formalnych w informatyce jest założenie, że zachowanie systemu można opisać zbiorem dyskretnych akcji atomowych. W najogólniejszym podejściu zakłada się, że temporalny porządek w tym zbiorze akcji jest porządkiem częściowym. Jednakże, porządek częściowy jest równoważny zbiorowi wszystkich porządków całkowitych, które nie są z nim niesprzeczne. Okazuje się, że jeśli mamy do czynienia tylko z właściwościami bezpieczeństwa i żywotności, to nie traci się żadnej informacji przez zastąpienie częściowo uporządkowanego zbioru zdarzeń zbiorem wszystkich ciągów otrzymanych przez rozszerzenie porządku częściowego na porządek całkowity. Tak więc, zachowanie można uważać za ciąg akcji. Współbieżność przejawia się jako indeterminizm, tzn. jeśli dwie akcje są współbieżne, to zbiór możliwych zachowań zawiera ciągi, w których te akcje są wykonywane w dowolnym porządku.

Formalizm oparty na sekwencyjności może być niedostateczny do uwzględnienia innych właściwości dotyczących zachowania systemu, np. czy dwie akcje występują współbieżnie. O ile takie właściwości mogą być interesujące w wypadku analizy określonego systemu, to nie stwierdziłem, aby były istotne w specyfikacji systemu.

Metoda aksjomatu przejść specyfikuje akcje atomowe zawierające pojedyncze operacje. Czy można specyfikować operacje nie stwierdzając, jakie są akcje atomowe?

Metodę aksjomatu przejść można rozszerzyć tak, aby umożliwiła specyfikowanie operacji nieatomowych, tzn. operacji złożonych z nieokreślonej liczby akcji atomowych. Napisanie takiej specyfikacji jest łatwe, na przykład, na wydruku 1 z poprzedniej części artykułu wystarczy usunąć nawiasy kątowe. Jednakże nie jest tak łatwo precyzyjnie stwierdzić, jakie jest znaczenie takiej specyfikacji i jak można zweryfikować poprawność realizacji. Metodę aksjomatu przejść można rozszerzyć, w celu uwzględnienia operacji nieatomowych, przez wprowadzenie formalnych pojęć opisanych w [2].

Modularność i dekompozycja hierarchiczna

Czy można zdekomponować hierarchicznie specyfikację w metodzie aksjomatu przejść?

Istnieją dwa rodzaje dekompozycji hierarchicznych:
- dekompozycja wewnątrz jednego poziomu abstrakcji,
- przedstawienie systemu wyższego poziomu jako kilku składowych niższego poziomu.

Drugi rodzaj dekompozycji wiąże się ze zmianą rozdrobnienia (atomowości) – zwykle pojedyncza akcja atomowa jest dekomponowana na zbiór akcji niższego poziomu – co nie dotyczy dekompozycji pierwszego rodzaju.

Dekompozycja wewnątrz pojedynczego poziomu wiąże się z takim zorganizowaniem informacji zawartej w specyfikacji, aby można ją łatwiej zrozumieć. Przykładowo, można zdekomponować aksjomat przejść wyrażając go jako koniunkcję kilku relacji i opisując oddzielnie każdy element koniunkcji. Ten rodzaj dekompozycji należy do zagadnień projektowania języka i nie stanowi zasadniczego problemu logicznego.

Przedstawienie systemu wyższego poziomu jako złożenia systemów niższego poziomu oznacza zrealizowanie systemu wyższego poziomu jako złożonego systemu niższego poziomu. Realizacja jednego systemu za pomocą systemu niższego poziomu była już dość dokładnie omówiona w tym artykule.

dokończenie na s. 25

NIE DAJEMY RECEPT
SPRZEDAJEMY NARZĘDZIA

ELEKTRONIKA FILM KOMPUTER
Zakład Spółdzielni Pracy UNICUM

ul. Barska 3/20, 02-315 Warszawa
tel. 23-67-57, tlx 816955

OFERUJE USŁUGI W DZIEDZINIE KOMPUTERYZACJI PRZEDSIĘBIORSTW

Podejmuje się kompleksowej obsługi kontrahentów:

- **sprzedaż sprzętu mikrokomputerowego (kompletacja, dostawa, serwis gwarancyjny i pogwarancyjny)**
- **opracowanie oprogramowania użytkowego (wdrożenie, szkolenie personelu)**

Służymy Państwu:

- **doradztwem organizacyjnym**
- **projektowaniem, oprogramowaniem oraz wdrażaniem systemów dedykowanych dla konkretnego użytkownika**
- **opracowaniem unikalnych programów wraz z nadzorem autorskim**

Sprzedajemy profesjonalne narzędzia dla profesjonalistów.

EO/955/88

MACIEJ M. SYSŁO
Instytut Informatyki
Uniwersytet Wrocławski

POLSKIE TOWARZYSTWO INFORMATYCZNE

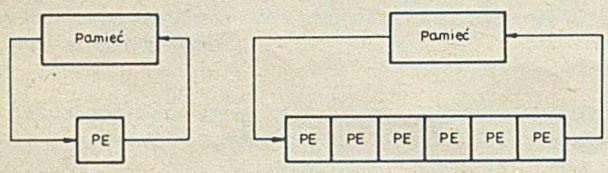
Maszyny i algorytmy równoległe (3)

Maszyny i algorytmy systoliczne

Maszyny systoliczne można uważać za sprzętowe realizacje algorytmów (obliczeń). Maszyna taka jest regularną siecią bardzo prostych procesorów *PE*, w której dane krążą w sposób rytmiczny. Określenie systoliczna pochodzi od angielskiego słowa *systole*, które jest terminem fizjologicznym oznaczającym rytmiczne skurcze serca i arterii powodujące pulsowanie krwi w ciele. Pierwsze maszyny systoliczne, będące realizacjami operacji macierzowych w technologii VLSI, zostały opracowane pod koniec lat siedemdziesiątych przez H. T. Kunga z Uniwersytetu Carnegie-Mellon. Komercyjne maszyny systoliczne pojawiły się na początku lat osiemdziesiątych.

Procesory w maszynach systolicznych wykonują najczęściej jedynie bardzo proste operacje. Informacje między procesorami przepływają w sposób potokowy, a komunikacja ze światem zewnętrznym odbywa się tylko za pomocą procesorów, które należą do brzegu układu. Zatem jedynie brzegowe procesory mogą być portami wejścia-wyjścia w czystosystolicznych układach. Maszyny systoliczne, w których dla świata zewnętrznego są dostępne także niektóre niebrzegowe procesory, będą nazwane pseudosystolicznymi.

Podstawowym zastosowaniem maszyn systolicznych są obliczenia, w których liczba operacji zdecydowanie przewyższa liczbę danych, a dane wielokrotnie występują w operacjach tego samego typu. Maszyny systoliczne zawdzięczają więc swoją dużą efektywność wielokrotnemu wykorzystaniu danych wprowadzonych do układu. Na rysunku 1 przedstawiono schemat maszyny systolicznej.



Rys. 1. Ewolucja maszyny klasycznej w systoliczną

Ocena efektywności algorytmów systolicznych, oprócz czasu obliczeń, obejmuje także oszacowanie całkowitej powierzchni zajmowanej przez układ VLSI realizujący dany algorytm. Z kolei oszacowanie powierzchni powinno uwzględniać przestrzeń potrzebną do zapisania wszystkich liczb występujących w obliczeniach oraz obszar przeznaczony na realizację procesorów *PE*.

OBLICZENIA MACIERZOWE

Omówię dwa rodzaje obliczeń macierzowych: obliczanie splotów i mnożenie dwóch macierzy.

Obliczanie splotów

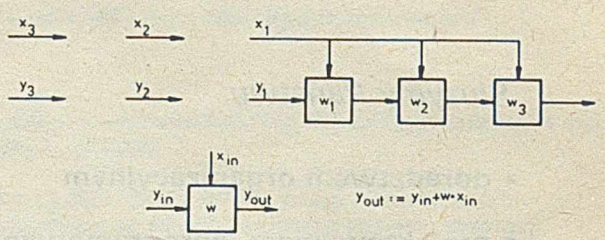
Klasycznym przykładem algorytmu systolicznego jest metoda obliczania splotów (ang. *convolution*). Niech będzie dany ciąg wag $w_1, w_2, \ldots, w_k$. Dla ciągu danych wejściowych $x_1, x_2, x_3, \ldots, x_n$ należy wyznaczyć ciąg $y_1, y_2, \ldots, y_{n+1-k}$ zdefiniowany następująco

$$y_i = w_1x_i + w_2x_{i+1} + \ldots + w_kx_{i+k-1}$$

Obliczanie splotów pojawia się w bardzo wielu standardowych podprogramach numerycznych, takich jak: dopasowywanie wzorca, korelacja, interpolacja, obliczanie wartości wielomianów, dyskretna transformacja Fouriera, mnożenie i dzielenie wielomianów. Poniżej opisano dwa algorytmy pseudosystoliczne *C1, C2*, a następnie – dwa algorytmy czystosystoliczne *C3, C4*. Dla większej przejrzystości prezentacji, opisami algorytmów są przykładowe sieci systoliczne dla $k = 3$.

Algorytm C1
Rozsyłanie danych: ruch wyników, wagi w miejscu.

W algorytmie systolicznym, którego schemat przedstawiono na rys. 2, wagi są umieszczane w procesorach przed rozpoczęciem obliczeń, częściowe wartości splotów y_j poruszają się od lewej do prawej, a kolejne elementy x_i są rozsyłane (ang. *broadcast*) jednocześnie do wszystkich procesorów. W pierwszym cyklu y_1, o początkowej wartości 0, przechodzi przez pierwszy procesor i kumuluje wartość w_1x_1. W tym cyklu wartości w_2x_1 i w_3x_1 nie są kumulowane przez żaden y_j. W drugim cyklu wartości w_1x_2 oraz w_2x_2 są kumulowane odpowiednio przez y_2 i y_1. Począwszy od trzeciego cyklu, poprawne wartości $y_1, y_2, \ldots$ zaczynają pojawiać się na wyjściu trzeciego procesora.



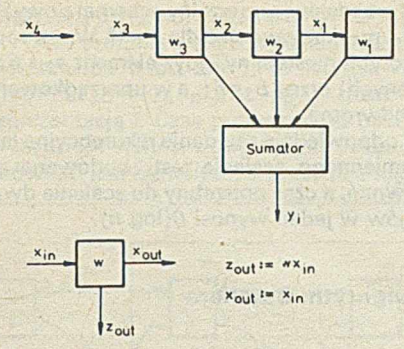
Rys. 2. Obliczanie splotów – systoliczny algorytm C1

Algorytm C2

Kumulacja wyników: ruch danych, wagi w miejscu.

Na rysunku 3 przedstawiono algorytm, w którym wszystkie składniki każdego splotu są obliczane równocześnie, a następnie wysyłane do sumatora, gdzie następuje ich kumulacja. Jeśli k jest stosunkowo duże, to sumator może mieć postać drzewa procesorów, w którym kolejne kumulacje są realizowane potokowo.

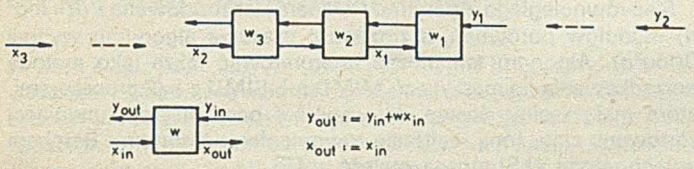
Przedstawione dwie metody obliczania splotów za pomocą algorytmów pseudosystolicznych korzystają z wielu mechanizmów uwspółbieżniania obliczeń, nie mają jednak wszystkich zalet układów czystosystolicznych. Na przykład, rozsyłanie danych i kumulacja wyników wymagają dodatkowych magistrali lub sieci globalnych. Dla dużych k może to powodować spowolnienie zegara układu. Przedstawione poniżej czystosystoliczne algorytmy nie korzystają z żadnej globalnej kumulacji i rozsyłania danych. Potencjalnie, bez większych trudności technicznych, takie układy systoliczne mogą być rozszerzane dla dowolnej liczby elementów.



Rys. 3. Obliczanie splotów – systoliczny algorytm C2

Algorytm C3
Wagi w miejscu, ruch danych i wyników z tą samą szybkością.

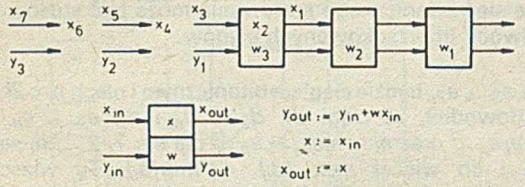
W algorytmie zilustrowanym na rys. 4 wagi są umieszczane w procesorach przed rozpoczęciem obliczeń, a dane i wyniki przepływają przez procesory w odstępach dwóch cykli między kolejnymi elementami; jest to słabą stroną tego rozwiązania, które tylko w połowie wykorzystuje moc procesorów, gdyż w przybliżeniu w każdym cyklu pracuje tylko połowa z nich.



Rys. 4. Obliczanie splotów – systoliczny algorytm C3

Algorytm C4
Wagi w miejscu, ruch danych i wyników.

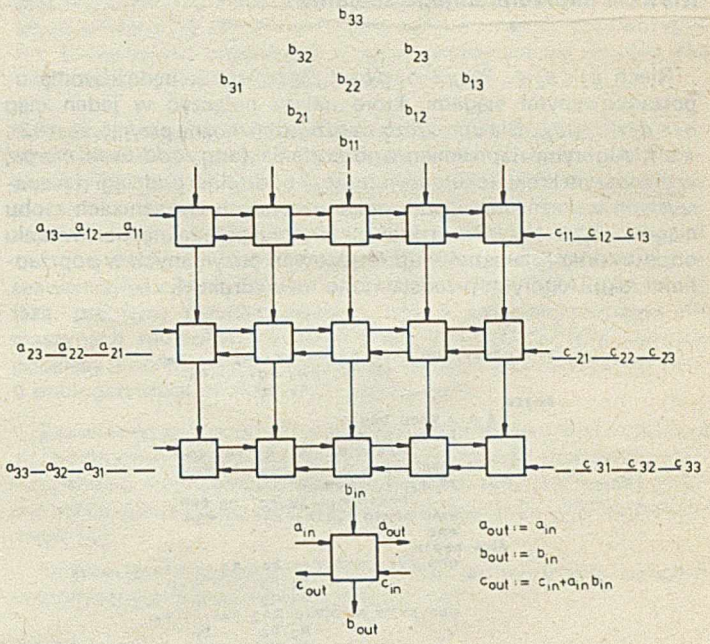
Algorytm przedstawiony na rys. 5 nie ma wady algorytmu C3, o której wspomniano w poprzednim zdaniu, dzięki przesyłaniu danych i wyników w tym samym kierunku, ale z różnymi szybkościami.



Rys. 5. Obliczanie splotów – systoliczny algorytm C4

Mnożenie dwóch macierzy

Najbardziej naturalną organizacją procesorów wykonujących systoliczne mnożenie dwóch macierzy stopnia n, $C = A \cdot B$, jest kwadratowa krata o boku n. Załóżmy, że elementy macierzy A poruszają się wierszami od lewej do prawej, elementy macierzy B poruszają się przekątnymi od góry do dołu, a elementy iloczynu C poruszają się wierszami od prawej do lewej. Ponieważ każdy element i-tego wiersza macierzy A występuje w rozwinięciu każdego elementu i-tego wiersza macierzy C, więc elementy macierzy A i C muszą być przedzielone jednym pustym cyklem, by żadne dwa elementy nie minęły się między procesorami (pusty cykl można zastąpić zerem otrzymując ten sam efekt). Nietrudno przedstawić układ, wyznaczający iloczyn dwóch macierzy, przy założeniu jednak, że elementy macierzy B z chwilą wejścia do układu pojawiają się we wszystkich procesorach danej kolumny. Takie rozwiązanie nie jest jednak zgodne z podstawowym założeniem o układach systolicznych, iż element wysłany przez jeden procesor osiąga inny połączony z nim bezpośrednio procesor z opóźnieniem przynajmniej 1. Układ taki może być jednak łatwo sprowadzony do układu czystosystolicznego przez opóźnienie każdego elementu macierzy B o 1 między kolejnymi procesorami i cofnięcie i-tych wierszy macierzy A i C o i-1 cykli. Tak zmodyfikowany układ jest czystosystoliczny (rys. 6).



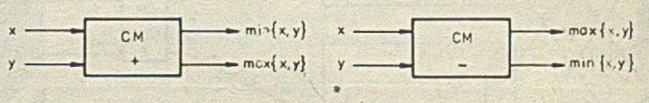
Rys. 6. Systoliczny algorytm mnożenia dwóch macierzy

Przedstawiony układ systoliczny mnoży dwie macierze stopnia n w czasie $O(n)$ i zajmuje powierzchnię $O(n^2 \log m)$, gdzie n^2 jest rzędem obszaru zajmowanego przez n^2 procesorów, a $\log m$ jest oszacowaniem górnym rozmiaru (tj. liczby bitów) liczb występujących w obliczeniach. Układ ten może być wykorzystany do mnożenia macierzy boolowskich stopnia n na powierzchni $O(n^2)$ i w tym samym czasie $O(n)$. Ten algorytm z kolei może być zastosowany do wyznaczania macierzy przechodniego domknięcia binarnej relacji (lub grafu) w czasie $O(n \log n)$. Wiele innych układów systolicznych dla problemów macierzowych i grafowych przedstawiono w książce [5].

Iloczyny macierzy szczególnych postaci mogą być realizowane za pomocą algorytmów systolicznych, w których struktura bezpośrednich połączeń między procesorami jest dostosowana do typu macierzy – czynników i iloczynu. Dla przykładu, iloczyn macierzy wstęgowej o szerokości wstęgi w przez wektor może być obliczony przez układ w procesorów połączonych liniowo, a do realizacji iloczynu dwóch macierzy wstęgowych (o szerokości wstęgi w) najodpowiedniejszy układ ma postać kwadratowej siatki o rozmiarach $w \times w$, w której każdy PE ma postać sześciokąta, tj. przepływają przez niego synchronicznie trzy strumienie danych.

SIECI SORTUJĄCE

Systoliczne algorytmy porządkowania ciągu liczb są powszechnie nazywane sieciami sortującymi (ang. *sorting network*). Podstawowym elementem sieci sortującej jest moduł porównania, który dla dwóch liczb na wejściu, podaje na wyjściu minimum i maksimum (rys. 7). Ze względu na swoją prostotę, moduły takie mogą być produkowane w technologii VLSI i tworzyć bardziej złożone układy.



Rys. 7. Moduły porównań

Poniżej opisano dwa algorytmy sortujące pochodzące od Batchera. Pierwszy z algorytmów jest oparty na innym algorytmie tego samego autora, łączącym dwa uporządkowane ciągi metodą naprze-

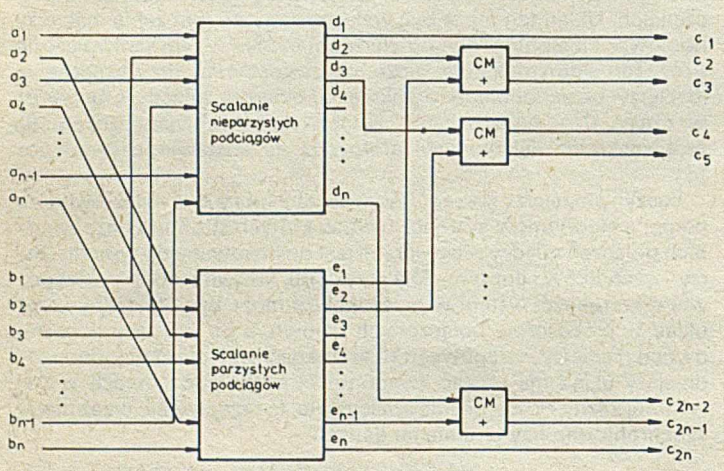
miennego scalania. Drugi algorytm jest oparty na właściwościach ciągów bitonicznych. Obie sortujące sieci Batchera są najszybszymi znanymi równoległymi algorytmami sortującymi.

Metoda naprzemiennego scalania

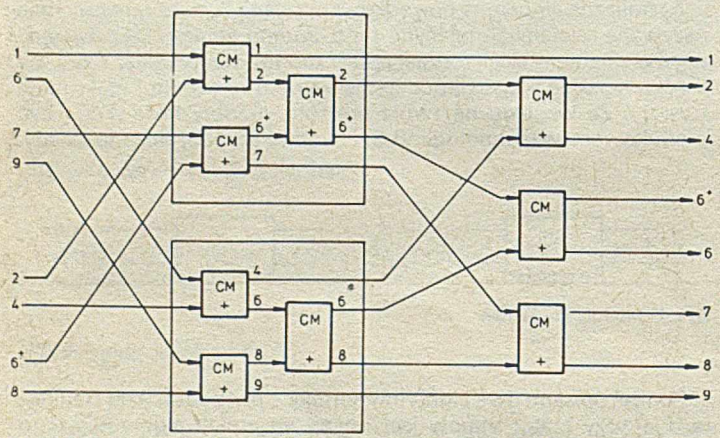
Niech $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$ i $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_n$, $n \geqslant 2$, będą dwoma uporządkowanymi ciągami, które należy połączyć w jeden ciąg $c_1 \leqslant c_2 \leqslant \ldots \leqslant c_{2n}$. Dla uproszczenia rozważań można przyjąć, że $n=2^k$, $k \geqslant 1$. Algorytm naprzemiennego scalania (ang. *odd-even merge*) w pierwszym kroku rekurencyjnie łączy oddzielnie podciągi o nieparzystych wskaźnikach i podciągi o parzystych wskaźnikach z obu ciągów $\{a_i\}$ i $\{b_i\}$. Drugi krok algorytmu polega na porównaniu odpowiednich elementów uporządkowań otrzymanych w poprzednim kroku. Algorytm przedstawiono na wydruku 1.

```
procedure ODD_EVEN_MERGE(n:a1,a2,...,an;
                         b1,b2,...,bn;
                         c1,c2,...,cn);
begin
    if n=2 then begin
        if a1<=b1 then
            begin d1:=a1; d2:=b1 end
        else begin d1:=b1; d2:=a1 end;
        if a2<=b2 then
            begin e1:=a2; e2:=b2 end
        else begin e1:=b2; e2:=a2 end
        end
    else begin
        ODD_EVEN_MERGE(n/2:a1,a3,...,an-1;
                       b1,b3,...,bn-1;
                       d1,d2,...,dn);
        ODD_EVEN_MERGE(n/2:a2,a4,...,an;
                       b2,b4,...,bn;
                       e1,e2,...,en)
    end;
    c1 := d1; c2n := en;
    for i:=1 to n-1 do begin
        c2i := min(di+1,ei);
        c2i+1 := max(di+1,ei)
    end
end
```

Wydruk 1. Algorytm naprzemiennego scalania



Rys. 8. Schemat działania algorytmu naprzemiennego scalania



Rys. 9. Przykład sortowania metodą naprzemiennego scalania

Na rysunku 8 przedstawiono ogólny schemat powyższego algorytmu, a na rys. 9 – przykładową sieć dla $n = 4$. Warto zauważyć, że na ogół jest to algorytm niestabilny, gdyż element $a_2 = 6$ występuje na początku (w danych) przed $b_3 = 6^+$, a w uporządkowanym ciągu ich kolejność jest odwrotna.

Rozwiązując odpowiednie równanie rekurencyjne łatwo obliczyć, że sieć naprzemiennego scalania jest zbudowana z $O(n \log n)$ modułów porównań, a czas potrzebny do scalenia dwóch uporządkowanych ciągów w jeden wynosi $O(\log n)$.

Równoległy algorytm Batchera

Powyższy algorytm można wykorzystać do skonstruowania sieci porządkującej dowolny ciąg liczb. Należy uporządkować ciąg $a_1, a_2, \ldots, a_n$, gdzie dla uproszczenia $n = 2^l > 0$. Równoległy algorytm Batchera wykonuje l kroków dla $i = 0, 1, \ldots, l-1$, a w każdym z nich stosując algorytm naprzemiennego scalania łączy wszystkie 2^{l-i-1} pary uporządkowanych podciągów długości 2^i (wydruk 2).

```
for i:=0 to l-1 do
    for 0<=j<=2^l-i-1 do simultaneously
        ODD_EVEN_MERGE(2^i:a[j2^i+1+1:j2^i+2^i];
                       a[j2^i+1+2^i+1:(j+1)2^i+1];
                       a[j2^i+1+1:(j+1)2^i+1])
{a[g:h] oznacza ciąg ag, ag+1, ..., ah}
```

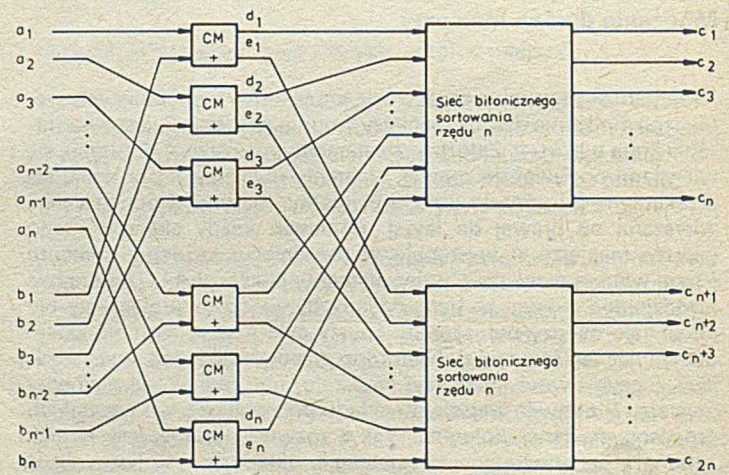
Wydruk 2. Algorytm Batchera

Sieć równoległego algorytmu Batchera jest zbudowana z $O(n \log^2 n)$ modułów porównań, a złożoność czasowa algorytmu wynosi $O(\log^2 n)$. Algorytm ten można interpretować także jako metodę porządkowania na maszynach MIMD lub SIMD z $n/2$ procesorami, które mają wolny dostęp do wspólnej pamięci przechowującej sortowany ciąg. Inną realizację równoległego algorytmu Batchera w technologii VLSI można znaleźć w [5].

Bitoniczne sortowanie

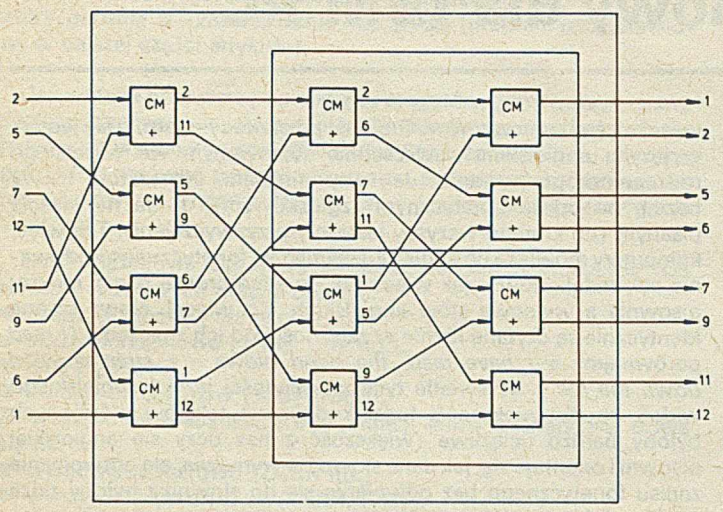
Wraz z poprzednimi algorytmami sortującymi, Batcher zaproponował jeszcze inny schemat porządkowania liczb, zwany siecią bitonicznego sortowania. Ciąg liczb $a_1, a_2, \ldots, a_n$ *jest bitoniczny (ang. bitonic)*, jeśli składa się co najwyżej z dwóch podciągów monotonicznych, tj. jeśli istnieje $j, 1 \leqslant j \leqslant n$, takie, że $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_j \geqslant a_{j+1} \geqslant \ldots \geqslant a_n$. Sieć bitonicznego sortowania porządkuje ciąg bitoniczny. Dla $j = 1$ lub $j = n$ ciąg bitoniczny jest monotoniczny. Każdy podciąg ciągu bitonicznego jest również bitoniczny, a także jeśli $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$ i $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_m$, to ciąg $(a_1, a_2, \ldots, a_n, b_m, \ldots, b_1)$ jest bitoniczny. Wynika stąd, że sieć bitonicznego sortowania może być stosowana do scalania dwóch uporządkowanych ciągów.

Niech $a_1, a_2, \ldots, a_{2n}$ będzie ciągiem bitonicznym i niech $n = 2^k$, $k \geqslant 0$. Batcher udowodnił, że ciągi $d_1, d_2, \ldots, d_n$ i $e_1, e_2, \ldots, e_n$, gdzie $d_i = min\{a_i, a_{n+i}\}$ oraz $e_i = max\{a_1, a_{n+i}\}$ dla i = 1, 2, ..., n, są także bitoniczne i co więcej $max\{d_i\} \leqslant min\{e_i\}$. Tę właściwość ciągów bitonicznych można rekurencyjnie zastosować do ciągów $d_1, d_2, \ldots, d_n$ oraz $e_1, e_2, \ldots, e_n$. W rezultacie, otrzymuje się algorytm bitonicznego sortowania, którego sieć przedstawiono schematycznie na rys. 10. Realizację tej sieci dla $2n = 8$, w której schemat



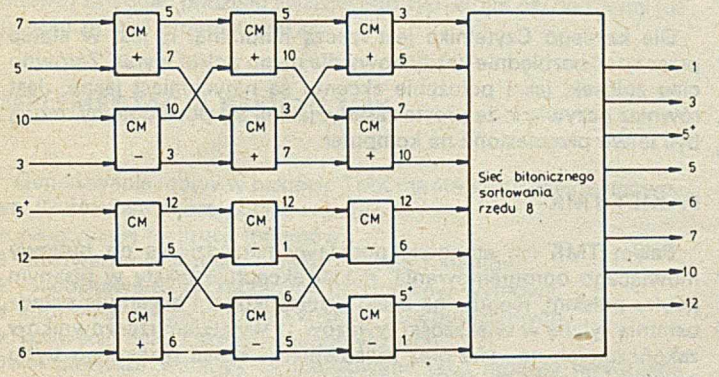
Rys. 10. Sieć bitonicznego sortowania rzędu 2n

połączeń między modułami porównań w dwóch sąsiednich kolumnach jest wyznaczony przez wartość funkcji SHUFFLE, przedstawiono na rys. 11. Jeśli przyjmie się, że wyjścia i wejścia modułów są oznaczone liczbami od 0 do $2n-1$, to wyjście i jest połączone z wejściem $SHUFFLE(i) = (i_{m-1}, i_{m-2}, \ldots, i_0, i_m)$, gdzie $i_m, i_{m-1}, \ldots, i_0$ jest dwójkową reprezentacją i. Czas potrzebny do posortowania ciągu bitonicznego rzędu $2n$ wynosi $O(\log n)$, zaś liczba zastosowanych modułów porównań jest ograniczona przez $O(n \log n)$.



Rys. 11. Przykładowa sieć bitonicznego sortowania rzędu 8

Jeśli ciąg $a_1, a_2, \ldots, a_n$, $n = 2^k (k \geq 0)$ nie jest bitoniczny, to można zastosować klasyczną metodę sortowania przez scalanie, realizowane przy użyciu sieci bitonicznego sortowania odpowiedniego rzędu. Dokładniej, w i-tym kroku dla $i = 0, 1, \ldots, k-1$ należy zastosować 2^{k-i-1} sieci sortujących rzędu 2^{i+1}. Na rysunku 12 przedstawiono sieć bitonicznego sortowania dowolnego ciągu 8-elementowego. Czas potrzebny do posortowania n liczb jest ograniczony przez $O \log^2 n$, a liczba użytych modułów porównań wynosi $O(n \log^2 n)$.



Rys. 12. Przykładowa sieć bitonicznego sortowania dowolnego ciągu długości 8

. . .

Dla dwóch podstawowych problemów rozpatrywanych w artykule, obliczenia iloczynu macierzy i porządkowania ciągu liczb, podano algorytmy równoległe, których złożoność obliczeniowa jest polilogarytmiczna, tzn. jest ograniczona przez wielomian zmiennej $\log n$, gdzie n jest rozmiarem problemu. Bardzo wiele problemów rozwiązywanych sekwencyjnymi algorytmami wielomianowymi, tj. należących do klasy P, można rozwiązywać polilogarytmicznymi algorytmami równoległymi na maszynach z wielomianową liczbą procesorów. Problemy te uważa się za najłatwiejsze w P. Z drugiej strony, klasa P zawiera problemy, których najprawdopodobniej nie można rozwiązywać w czasie polilogarytmicznym na równoległej maszynie. Należą do nich: programowanie liniowe, wyznaczanie maksymalnego przepływu w sieciach i porządkowanie wierzchołków grafu zgodnie z metodą przeglądania w głąb. Problemy te należą do klasy problemów P-zupełnych, tzn. każdy inny problem z P może być przetransformowany na problem P-zupełny z wykorzystaniem logarytmicznej pamięci. Oznaczmy przez POLYLOGSPACE klasę problemów rozwiązywalnych algorytmami o polilogarytmicznej złożoności pamięciowej. Jeśli którykolwiek z problemów P-zupełnych jest w tej klasie, to $P \subseteq$ POLYLOGSPACE, co uważa się za mało prawdopodobną ewentualność. Problemy P-zupełne są zaliczane do najtrudniejszych problemów wielomianowych, są one z natury sekwencyjne. Ich status wśród problemów klasy P porównuje się do roli, jaką odgrywają problemy NP-zupełne w klasie NP.

Przytoczę jeszcze hipotezę, zwaną *tezą o równoległych obliczeniach*, która odgrywa podstawową rolę w teorii złożoności obliczeniowej. W myśl tej tezy, dla każdej funkcji T zmiennej rozmiaru problemu n, klasa problemów rozwiązywalnych na maszynie równoległej o nieograniczonej współbieżności w czasie wielomianowym zmiennej $T(n)$ jest klasą problemów rozwiązywalnych na maszynie sekwencyjnej z pamięcią wielomianową zmiennej $T(n)$. Zgodnie z tą tezą, problemy rozwiązywalne w czasie polilogarytmicznym na maszynach równoległych (tj. z klasy POLYLOGSPACE) tworzą podklasę problemów rozwiązywalnych algorytmami sekwencyjnymi o polilogarytmicznej złożoności pamięciowej.

Zauważmy, że algorytmy równoległe dla maszyn z wielomianową liczbą procesorów nie są w stanie zmienić statusu problemów NP-zupełnych, oczekiwać jednak można, że maszyny równoległe zwiększą efektywność algorytmów przeliczeniowych dla tej klasy problemów [3].

Złożoności obliczeniowej problemów w modelach obliczeń współbieżnych poświęcone są prace [1–4].

LITERATURA

[1] Coffman E. G., Jr., Denning P. J.: Operating Systems Theory. Prentice-Hall, Englewood Cliffs (NJ), 1973

[2] Gruska J.: Systolic Computations, Springer-Verlag, Berlin, 1985

[3] Kindervater G. A. P., Lenstra J. K.: An Introduction to Parallelism in Combinatorial Optimization. Discrete Appl. Math., Vol. 14, pp. 135–156, 1986

[4] Reif J. H.: Depth-first Search Is Inherently Sequential. Information Processing Letters, Vol. 20, pp. 229–234, 1985

[5] Ullman J. D.: Computational Aspects of VLSI. Computer Science Press, 1984.

MIECZYSŁAW A. KŁOPOTEK
Instytut Podstaw Informatyki
Polska Akademia Nauk
Warszawa

TMK – pakiet syntezy mowy dla IBM PC

W artykule omówiono zagadnienia komputerowej generacji mowy z tekstu pisanego i przedstawiono opis realizacji pakietu syntezy mowy TMK, przeznaczonego dla mikrokomputerów typu IBM PC XT/AT.

Prace nad syntetyczną generacją mowy zostały rozpoczęte z górą 20 lat temu. Dziś syntezatory mowy są nie tylko produktami laboratoryjnymi, lecz pojawiły się również na rynku. W Japonii, na przykład, można kupić zegarek podający godzinę głosem męskim lub kobiecym. Opracowano też kilka programów na mikrokomputery, które generują mowę na podstawie notacji fonetycznej. Stworzono nie tylko systemy dla języka angielskiego (*NRL, MITalk* [2]), ale także dla innych języków (np. greckiego [8] i węgierskiego [3]).

Potencjalnymi dziedzinami zastosowań mówiącego oprogramowania są:
- komputerowe wspomaganie nauczania (jako składnik automatycznego kontrolera dyktand, jako „wykładowca" dla ucznia o słuchowym typie przyswajania wiedzy itp.),
- poczta elektroniczna,
- systemy zapytań i odpowiedzi,
- podsystemy ostrzegawcze i informujące (przywołanie użytkownika w systemie wielozadaniowym, wywołanie operatora z hali produkcyjnej, słowne powtórzenie wyświetlanej informacji i in.),
- edytory tekstów (czytanie na głos fragmentów tekstu),
- pomoc dla niewidomych i słabo widzących,
- oprogramowanie systemowe dla niewidomych programistów.

Analizując możliwości wykorzystania oprogramowania tego rodzaju możemy dojść do wniosku, że brak pola zastosowań nie jest główną przeszkodą w ich upowszechnieniu. Wydaje się natomiast, że jest nią powszechna tendencja do posługiwania się językiem angielskim w informatyce. Język angielski nie nadaje się bowiem do generowania mowy z tekstu pisanego. Wspomniany *MITalk* posługuje się słownikiem morfów o 8000 elementach niezbędnych do dekompozycji słowa oraz dokonuje częściowej analizy syntaktycznej. Okrojona wersja mikrokomputerowa tego systemu zajmuje 104 KB pamięci. Do generacji mowy nadaje się natomiast język polski. Z tego względu podjęliśmy się opracowania pakietu TMK (skrót od: „Tu Mówi Komputer") generującego akcentowaną mowę z przedłożonego tekstu w języku polskim.

SYNTEZA MOWY Z TEKSTU PISANEGO

Mowa ludzka jest, jak wiadomo, generowana przez drgające struny głosowe, przy czym zgłoski, słowa i zdania to ciąg dźwięków różniących się widmem amplitudowo-częstotliwościowym. Zatem, w uproszczeniu zadanie generacji mowy sprowadza się do identyfikacji sekwencji widm odpowiadających poszczególnym zgłoskom i ich syntezie w syntezatorach dźwięku. Dotychczas opracowano, a następnie udoskonalono szereg syntezatorów mowy [6]. Celem tych prób jest także usprawnienie procesu syntezy, aby generowana mowa była coraz bardziej zbliżona do widma wyjściowego (ludzkiego) [5]. Obecnie na rynkach zachodnich są dostępne gotowe syntezatory mowy sterowane zgłoskami (w cenie około 1000 marek RFN).

Oczywistą przeszkodą w upowszechnianiu się systemów mówiących o odpowiednio wysokiej jakości generowanej mowy jest wysoka cena syntezatorów. Głównej przeszkody upatruje się jednak w języku angielskim, powszechnie stosowanym we wszelkiego rodzaju oprogramowaniu. Jak wspomniałem, syntezatory mowy bazują na opisie fonetycznym (zgłoskowym), a nie na tekście pisanym (do którego wszyscy jesteśmy przyzwyczajeni). Tak więc, komputery mówiące po angielsku wymagają fonetycznego kodowania informacji, gdyż – jak wiadomo – nie ma prostej relacji między pisownią a wymową słów angielskich. Co więcej, słowa pisane identycznie są czytane różnie w zależności od ich znaczenia (*I read* porównajmy z *I have read, the wind blows* – z *streams wind down the hill*). W świetle tych właściwości języka angielskiego widać, że dla większości ludzi kodowanie tekstów mówionych byłoby bardzo uciążliwe (większość z nas uczy się angielskiej pisowni i orientuje się, jak dane słowo się wymawia, ale odtworzenie zapisu fonetycznego bez odwołania się do słownika byłoby poza możliwościami ogółu programistów.

Mogłoby się wydawać, iż język rosyjski nadaje się do generowania mowy z tekstu pisanego. Jednakże należy zauważyć, że – pominąwszy takie wyrazy jak мука, które w zależności od akcentu znaczą co innego (mąka lub męka) – generalnie o tym, w jaki sposób czyta się literę *o*, decyduje akcent, którego położenie trudno ustalić bez obszernego słownika.

Wymowa wielu sekwencji literowych oraz akcentowanie sylab w języku niemieckim zależy od kompozycji słów złożonych. Również język niemiecki wymaga słownika, jeśli mowa ma być generowana z tekstu pisanego.

Dla każdego Czytelnika jest rzeczą naturalną, iż jest w stanie przeczytać bezbłędnie każdy nowy dla niego polski wyraz. Zarówno ciąg zgłosek, jak i położenie akcentu są natychmiast jasne. Jest również oczywiste, że proste zasady, jakimi się posługujemy, mogą być łatwo przeniesione na komputer.

PAKIET TMK

Pakiet TMK ma stanowić podstawy narzędziowe do budowy mówiącego oprogramowania. Pakiet akceptuje teksty w pisanym języku polskim, reaguje na znaki przestankowe i akcentuje przedostatnią sylabę w większości wyrazów (z wyjątkiem rzeczowników zakończonych na *-yka* oraz czasowników czasu przeszłego trybu oznajmującego i przypuszczającego).

Na pakiet składają się:

- **moduły syntetyzujące mowę:**
  niemowa.exe – moduł inicjujący,
  gadaj.exe – moduł generacji mowy z polecenia systemowego,
  czytaj.exe – moduł odczytujący przygotowany tekst,
  msc_gadaj.obj – moduł przyłączany do modułów aplikacyjnych (generuje mowę z ciągu znaków będących jego parametrem),
  ostrzegaj.obj – moduł przyłączany do modułów aplikacyjnych (zezwala na generowanie ostrzeżeń ze specjalnej struktury ***msg*, odwołując się do niej poprzez numer komunikatu),**

- **moduły polskich liter:**
  klawisz.exe – przeprogramowanie klawiatury (małe polskie litery: *ALT-litera_łacińska,* duże polskie litery: *CTRL-litera_łacińska*),
  poldruk.exe – program drukowania po polsku,

- **moduły pomocnicze:**
  liczba.obj – konwersja ciągu cyfr na słowny odpowiednik,
  data.obj – konwersja ciągu cyfr na słowną datę,
  dobrze_czytaj.exe – konwersja tekstu pisanego zawierającego skró-

ty nazw, liczby, daty i godziny,
skróty.exe – program przygotowania listy skrótów,
msc_tempo – program regulowania szybkości mowy.

- **moduły demonstracyjne:**

demo.bat – przykłady zastosowania modułu *gadaj.exe*
referat.bat – przykład zastosowania *czytaj.exe*
wiadomości.c – przykład zastosowania *ostrzegaj.obj*,
która_godzina.c – zastosowanie *msc_gadaj* (program przedstawiony w dalszej części artykułu).

Jak widać z powyższego przeglądu modułów, generowanie mowy z tekstu pisanego nie jest zadaniem banalnym, sprowadzającym się jedynie do wygenerowania dźwięków. Stworzenie pakietu wymaga pokonania zarówno trudności formalno-technicznych (wrowadzanie tekstów w języku polskim), jak i językowych (identyfikacja skrótów).

Skróty językowe dzielą się na liczbowe oraz alfabetyczne. Do skrótów liczbowych można zaliczyć:
- liczby całkowite i ułamkowe,
- liczby porządkowe (7-my, 9-ty itp.),
- daty i godziny,
- notacje przedziałów (1987–1989),

Ze względu na specyfikację interpretacji skróty alfabetyczne dzielą się na:
- wymawiane literowo (ZSRR itp.),
- wymawiane jako słowa (CEMI, PAN),
- wymawiane w pełnym brzmieniu (pon, np, dr).

O ile skróty liczbowe można zasadniczo przetworzyć na ich pełną pisownię algorytmicznie, to interpretacja skrótów alfabetycznych jest bardziej skomplikowana. Skróty wymawiane w pełnym brzmieniu muszą być zawarte w wyczerpującym słowniku.

Odnośnie pozostałych skrótów można zastosować pewne heurystyki pozwalające rozróżnić obie grupy (literową i słowną). Przykładowo, skrót złożony z samych spółgłosek jest na pewno wymawiany literowo (przy czym akcent pada na ostatnią, a nie przedostatnią sylabę: *pekao*, a nie *pekao* itp. Niektóre kombinacje literowe nie mogą być łatwo wymówione w ramach jednej sylaby. Na przykład, w przypadku kombinacji liter: *pk* potrafimy powiedzieć *sypki*, bo możemy dokonać podziału sylabowego *syp-ki*, ale *pki* będziemy już uważać za skrót literowy *pekai*.

IMPLEMENTACJA PAKIETU TMK

Generowanie mowy w pakiecie TMK ma strukturę wielowarstwową. Wyróżniamy następujące warstwy:
- fonetycznej realizacji fonemów,
- przetwarzania słowa pisanego na ciąg fonemów,
- rozpoznawania szczególnych przypadków wymowy,
- kontroli szybkości mowy i znaków interpunkcyjnych,
- konwersji liczb i skrótów alfabetycznych.

Dwie pierwsze warstwy są realizowane jako moduły rezydujące w pamięci (ok. 45 KB), a pozostałe jako samodzielne moduły wywoływane w poleceniach systemowych lub jako podprogramy dołączane do programów użytkownika. Dwie pierwsze warstwy są niedostępne dla użytkownika. Warstwy wywołują się nawzajem w kolejności odwrotnej do podanej.

Konwencja kodowania polskich liter

Litery polskie, które nie należą do alfabetu łacińskiego, są kodowane jednolicie jako dwuznakowe ciągi złożone ze źródłowej litery łacińskiej i znaku „Λ" (dotyczy to liter *ą, ę, ć, ń, ś, ź, ó, ł, Ą, Ę, Ć, Ń, Ś, Ź i Ł*) lub litery łacińskiej i znaku „ⓐ" (*ż* i *Ż*). Celem ułatwienia stosowania tego kodu przygotowano moduł **KLAWISZ.EXE**, który w otoczeniu programu *ANSI.SYS* przedefiniowuje klawiaturę tak, by kombinacja klawiszy *ALT-litera* dawała sekwencję *mała_litera* i „Λ", kombinacja *CTRL-litera* dawała sekwencję *duża_litera* i „Λ", natomiast *ALT-X* – „*z* ⓐ " i *CTRL-X* – „*Z* ⓐ ". Ponadto, dla plików napisanych w konwencji TMK program

POLDRUK *długość_strony lewy_margines nazwa_pliku*

drukuje plik tekstowy, podstawiając litery polskie zamiast odpowiednich sekwencji i zatrzymując się co *długość_strony* (możliwość ustawienia papieru).

Tekst przygotowany w konwencji TMK może być bezpośrednio czytany na głos za pomocą programu

CZYTAJ *nazwa_pliku*

Tekst ten może zawierać sekwencje sterujące wysokości; dźwięku mowy (a zarazem szybkością mowy, wiążąc wyższy ton z szybszym tempem). Sekwencje te mają postać:

$Mliczba,

gdzie *liczba* jest nieujemną liczbą całkowitą (rosnąca jej wartość oznacza spadającą wysokość dźwięku).

Generowanie mowy

Istnieje wiele technik kodowania i związanych z nimi technik generowania sygnałów mowy. Dzieli się je na dwie podstawowe klasy:
- kodowanie kształtu sygnału,
- kodowanie oparte na sposobie generacji mowy przez człowieka (kodowanie parametryczne).

Metody pierwszej klasy są znacznie łatwiejsze do zaprogramowania i wymagają mniej kosztownych urządzeń zewnętrznych. Dlatego nasz wybór padł na pierwszą klasę. Wadą tych technik jest jednak wzajemne uzależnienie parametrów mowy (np. wysokości tonu od szybkości mowy).

Do technik kodowania kształtu sygnału należą:
- PCM – impulsowa modulacja kodowa (ang. *Pulse Code Modulation*),
- log PCM – logarytmiczna PCM,
- DPCM – różnicowa PCM (ang. *Differential PCM*),
- DM – modulacja delta (ang. *Delta Modulation*),
- ADPCM – adaptacyjna DPCM (ang. *Adaptive DPCM*),
- ADM – adaptacyjna modulacja delta (ang. *Adaptive Delta Modulation*),

Technika PCM bazuje na twierdzeniu o próbkowaniu Nyquista [1] głoszącym, że sygnał jest jednoznacznie określony przez ciąg próbek pobieranych w odstępach czasu

$$r = \frac{1}{2W}$$

dla sygnału idealnego o widmie częstotliwości ograniczonym do pasma $(-W, +W)$. Sygnał oryginalny można bowiem odtworzyć z takiego ciągu próbek za pomocą funkcji interpolacyjnych o postaci

$$\frac{\sin x}{x}$$

Wprawdzie sygnały nie są sygnałami idealnymi, ale część słyszalna jest w rzeczywistości ograniczona do $W=20$ kHz, a ograniczenie do $W=5$ kHz daje w praktyce zadowalające efekty. Tak więc, sygnał mowy jest próbkowany w zadanych odstępach czasu, a następnie uzyskana wartość amplitudy jest kwantowana, dając cyfrowy zapis sygnału mowy. Generowanie mowy polega na odwróceniu tego procesu.

Technika DPCM jest pomyślana jako modyfikacja techniki PCM w celu zmniejszenia wymagań pamięciowych. Stwierdzono, że przy zadanej dużej częstości próbkowania amplituda sygnału zmienia się stosunkowo powoli, a zatem zamiast każdorazowo pamiętać bezwzględną wartość amplitudy można pamiętać (z wykorzystaniem mniejszej liczby bitów) różnicę amplitud. Generowanie wymagałoby pamiętania poprzedniej wartości amplitudy. Jeżeli częstotliwość próbkowania jest odpowiednio mała, to zmiany można kodować z wykorzystaniem jednego bitu – jest to technika DM.

Inną metodą zmniejszenia pamięci jest wykorzystanie faktu, iż ludzkie ucho reaguje na dźwięk z logarytmiczną skalą stopniowania.

W technice log PCM pamięta się logarytm amplitudy, a nie samą amplitudę.

W metodzie ADPCM wykorzystuje się poprzednie próbki celem uwzględnienia zmian sygnału mowy. Jeżeli zmiany są szybkie, to stosuje się mały odstęp między próbkami, jeżeli powolne – duży.

Do klasy kodowania parametrycznego należą:
- kodowanie formantowe (szerzej omówione w [7]),
- kodowanie kanałowe [4],
- predykcyjne kodowanie liniowe LPC (ang. *Linear Predictive Coding*) [5].

Pakiet TMK jest przeznaczony dla mikrokomputerów typu IBM PC XT/AT. Z naszego punktu widzenia głośnik instalowany w tych komputerach charakteryzuje się dwoma interesującymi stanami: włączony i wyłączony (wykorzystanie pełnej gamy możliwych wysokości tonu spowolniałoby nadmierne działanie generatora; nie jest znany sposób regulacji amplitudy dźwięku). Dlatego posłużyliśmy się techniką czasowej modulacji impulsów o małej szerokości MCIMS [1]. Polega ona, mówiąc obrazowo, na tym, iż wyższa amplituda odzwierciedlanego sygnału odpowiada większej częstotliwości krótkich impulsów o jednakowych amplitudach. Inercja układu nerwowego człowieka powoduje, że słyszy on amplitudę efektywną (uśrednioną w pewnym przedziale czasu). Istnieje zatem tutaj pewna analogia do kodowania metodą PCM.

Regulacji wysokości dźwięku mowy (szybkości mowy) dokonuje się przez wydłużanie i skracanie czasu między przetwarzaniem kolejnych bitów, oznaczających obecność lub nieobecność impulsu o małej szerokości w danym wycinku czasu. Ta regulacja wysokości mowy jest istotna przy przenoszeniu pakietu z IBM PC, ponieważ metoda generowania mowy nie pozwala na bieżący odczyt czasu rzeczywistego i dostosowanie na bieżąco tempa mowy do faktycznego upływu czasu (spowodowałoby to zmniejszenie szybkości programu, a przez to częstotliwości dźwięku mowy poniżej poziomu słyszalności). Należy więc samodzielnie przetestować ustawienie wysokości tonu wywołując podprogram *msc_tempo* z różnymi wartościami parametru lub wykorzystując sekwencję sterującą *$M* w tekście mówionym.

Przetwarzanie słowa pisanego na ciąg fonemów

Zadaniem tej warstwy jest podzielenie słowa na odcinki odpowiadające polskim dźwiękom oraz określenie położenia akcentu. Przykładowo słowo *chrabąszcz* zostanie podzielone na *ch-r-a-b-ą-sz-cz* i przekazane warstwie generowania zaznaczeniem pozycji akcentu. Podziału fonetycznego dokonuje się według słownika jednostek fonetycznych, natomiast pozycję akcentu ustala się odliczając samogłoski fonetyczne. Słowa o akcencie nietypowym (matematyka, chcieliście, należelibyśmy itp.) identyfikuje się przez słownik końcówek. Ponieważ program tej warstwy rezyduje w pamięci, więc ze względów oszczędnościowych przetwarzanie słów nietypowych (np. *marznąć*), w których sekwencje liter mają odmienną niż zwykle interpretację zgłoskową, przesunięto do warstwy wyższej. Na tym poziomie zarezerwowano specjalną literę θ separującą oddzielnie wymawiane litery (np. *marznąć* koduje się dla tego poziomu jako *marθznąć*).

Rozpoznawanie szczególnych przypadków mowy

Zadaniem tego poziomu jest zapewnienie poprawnej wymowy wyrazów o wymowie odmiennej niż wynika to z ogólnych zasad wymowy. Jedną grupę stanowi słowo *marznąć* oraz jego wyrazy pochodne. Opracowano następującą regułę heurystyczną identyfikacji słów tej grupy:
– znaleźć literę *r*,
– stwierdzić, czy lewym kontekstem jest *ma* ; jeżeli tak, to sprawdzić czy prawym kontekstem jest *zX*, gdzie *X* oznacza dowolną spółgłoskę lub literę ⓐ ; jeżeli tak, to podstawić za literę *r* ciąg *rθ* .

Druga grupa słów o odmiennym sposobie odczytu dotyczy zmiękczającego znaczenia litery *i* w złożeniach *si* , *ci* , *zi* , *ni*. Przykładami wyjątków są grupy słów *sinus* i *silikon* . W tym wypadku:
– poszukuje się litery *i* ,
– sprawdza lewy kontekst *s*,
– *sprawdza prawy kontekst nus , lik*,
– w pomyślnym przypadku podstawia za *i* ciąg *θi*.

Kontrola szybkości mowy i znaków interpunkcyjnych

Jak już wspomniano, TMK umożliwia sterowanie szybkością i wysokością mowy. Służy do tego celu moduł *msc_tempo()* oraz sekwencja sterująca *$M*. W ten sposób można osiągnąć efekt dialogu w toku narracji. Na przykład w wyniku przygotowania poniższego tekstu do odczytania przez moduł CZYTAJ.EXE:
$M2 – Przyszła baba do lekarza, a lekarz:
$M3 – Co Pani jest?
$M1 – Krawcowa.
narrator będzie mówić tonem wyższym od lekarza, a niższym od pacjentki.

Spacje w tekście powodują generowanie krótkiego odstępu czasu, a znaki – kropki, wykrzyknika itp. – dłuższego. Sterowanie szybkością mowy ma charakter globalny, tzn. tempo ustawione w programie użytkowym lub w module pakietu TMK jest zachowane do najbliższej zmiany.

Konwersja liczb i skrótów

Zaimplementowano najprostszą formę przetwarzania skrótów, mianowicie proste szukanie w słowniku i podstawianie odpowiedniego słowa lub słów zamiast znalezionego skrótu. Jeśli zostanie napotkany ciąg znakowy rozpoczynający się cyfrą lub znakiem podkreślenia (nie występującym w słowniku), to przeprowadza się identyfikację ciągu znakowego:
– jeśli zawiera on wyłącznie cyfry, to jest to liczba całkowita,
– jeśli zawiera znak %, to jest to liczba typu *ilo* z następującą po niej frazą *procentowy*,
– jeśli zawiera jedną kropkę, po której występują dwie cyfry, jest to godzina,
– jeśli zawiera dwie kropki oddzielone jedną lub dwiema cyframi, jest to data itd.
Po sklasyfikowaniu ciągu znakowego zamienia się go na odpowiedni ciąg słów. W aktualnej wersji pakietu nie dokonuje się pełnej kontroli przynależności ciągu do klasy. W przyszłości trzeba będzie jednak uwzględnić informacje kontekstowe celem wyeliminowania fałszywych klasyfikacji (należy, na przykład, odróżnić napisy *2 dzieci, 2 kobiety, 2 mężczyzn* itp.).

```
static char miesiąc[] =
{ "stycznia", "lutego", "marca",..... "listopada", "grudnia"};
main()
{ int dz,mc,rok,godz,min; char str [180];
  splittime ( &rok,&mc,&dz,&godz,&min );
  /* funkcja podaje aktualny czas zegarowy */
  msc_gadaj ( "Mamy dziś dzień" );
  /* msc_gadaj jest dostarczany w module msc_gadaj.obj */
  liczba ( 'P', dz, str, "y" );
  /* funkcja dostarczana przez moduł liczba.obj,
  . * parametr wejściowy 'P' oznacza generowanie liczb
  . *    porządkowych,
  . * parametr wejściowy "y" to końcówka dla liczb porządkowych,
  . * parametr wejściowy dz to liczba , jaką się zamienia
  . *    na ciąg słów i zwraca jako wartość parametru str.
  . */
  msc_gadaj (str);
  msc_gadaj (miesiąc [mc-1]);
  liczba ('P', rok, str, "ego"); msc_gadaj(str);
  msc_gadaj ("roku, mamy godzinę");
  liczba ('P', godz, str, "a"); msc_gadaj(str);
  liczba ('G', min, str);       msc_gadaj(str);
  /* 'G' oznacza generowanie liczebnika głównego */
```

Tekst źródłowy programu podającego „na głos" datę i godzinę

dokończenie na s. 28

ZBIGNIEW STRUK
EWA ZABŻA-TARKA
Instytut Informacji Naukowej, Technicznej
i Ekonomicznej
Warszawa

Prosty system zarządzania bibliograficzną bazą danych dla mikrokomputera IBM PC/XT

W ośrodkach informacji naukowej i bibliotekach często trzeba sporządzać różnego rodzaju zestawienia (bibliografie) lub wyszukiwać pewne informacje w nich zawarte. Jest to praca żmudna i uciążliwa, dlatego też celowe jest jej zautomatyzowanie.

W Instytucie Informacji Naukowej, Technicznej i Ekonomicznej zaprojektowano w 1987 r. mikrokomputerowy system bibliograficzny. Służy on nie tylko do przechowywania danych, lecz jest także przeznaczony do zakładania bibliograficznej bazy danych, aktualizowania rekordów w bazie oraz wyszukiwania odpowiednich danych. Jako odpowiedź użytkownik może otrzymać dane w postaci rekordów, spełniających zadany warunek, bądź pełny opis tych rekordów. Wyniki mogą być wyprowadzone na drukarkę albo wyświetlone na ekranie monitora.

System jest skonstruowany modułowo, dzięki czemu jest łatwy do modyfikowania i ewentualnego rozbudowania w przyszłości. Został napisany w języku dBASE III, a następnie skompilowany w celu zwiększenia szybkości działania. Jest to system konwersacyjny – użytkownik korzysta z niego wybierając odpowiednie możliwości z podanego menu lub wypełniając wyświetlony formularz. Posługuje się przy tym trzema zbiorami danych: bibliograficzną bazą danych, wykazem słów kluczowych oraz indeksem autorów, zawierającym nazwiska wszystkich autorów wraz z odpowiadającymi im numerami rekordów w bazie głównej.

MOŻLIWOŚCI SYSTEMU

Mikrokomputerowy system bibliograficzny składa się z kilkunastu procedur. Jest to system konwersacyjny, w którym użytkownik wybiera różne warianty z podanego menu. Główne menu umożliwia wybór spośród następujących funkcji:

- wprowadzenie nowych rekordów do bazy danych,
- wydrukowanie pełnej lub częściowej zawartości bazy danych oraz indeksu autorów i zestawu słów kluczowych,
- aktualizowanie danych w bazie,
- wyszukiwanie podstawowe (pytania proste),
- wyszukiwania złożone (pytania złożone),
- zmiana nagłówka,
- wyjście z systemu.

Po wybraniu odpowiedniej czynności system automatycznie przechodzi do jej wykonania. Każdą czynność można wykonywać wielokrotnie dla nowych parametrów. Jeżeli bibliograficzna baza danych nie została jeszcze założona, to dozwolone jest tylko wprowadzenie nowych rekordów i zmiana standardowego nagłówka, umieszczanego w raportach generowanych przez system.

Rekordy na wyjściu muszą być wyświetlane na ekranie monitora albo wydrukowane na drukarce. W tym ostatnim wypadku użytkownik może określić liczbę wykonywanych kopii. Na życzenie użytkownika zawartość rekordów może być drukowana z prawostronnym wyrównaniem tekstu (bardziej eleganckie, lecz jednocześnie nieco bardziej czasochłonne) albo bez takiego wyrównania – z łamaniem tekstu na spacji lub przecinku, bez rozsuwania słów w wierszu.

STRUKTURA REKORDU

Każdy rekord w bibliograficznej bazie danych składa się z następujących pól:

AUTOR – długość 110 znaków – zawiera nazwiska wszystkich autorów danej publikacji, np. Kowalski J., Zieliński B.;
TYTUŁ – 220 znaków – tytuł publikacji, np. „Mikrokomputerowy system zarządzania bazą danych";
MIEJSCE – 20 znaków – miejsce wydania publikacji, np. Warszawa;
WYDAWCA – 60 znaków – nazwa wydawcy, np. CINTE. Prace IINTE;
ROK – 4 znaki – rok wydania, np. 1986;
WOLUMEN – 20 znaków – numer wolumenu lub zeszytu, np. Vol. 4 No. 6;
STRONY – 20 znaków – liczba stron publikacji lub numer pierwszej i ostatniej strony, np. 15 lub 23–31;
JĘZYK – 15 znaków – nazwa języka, w jakim została napisana dana publikacja, np. pol., ang., ros., niem.;
KLUCZE – 140 znaków – lista słów kluczowych opisujących daną publikację, np. bazy danych, systemy informacyjno-wyszukiwawcze, mikrokomputery;
RODZAJ – 30 znaków – rodzaj publikacji, np. artykuł, zwarte, utwór, mat. konferencyjne, spr. naukowe lub norma.

WPROWADZANIE DANYCH

Funkcja wprowadzania danych umożliwia dołączanie w prosty sposób nowych rekordów do istniejącej już bazy danych lub założenie nowej bazy danych. Użytkownik otrzymuje wyświetlony na ekranie formularz o strukturze takiej, jak struktura rekordu z bazy. Zadaniem użytkownika jest wypełnienie wszystkich pól tego formularza i sprawdzenie ich poprawności. Jeśli użytkownik nie zna wartości niektórych pól, to pozostawia je puste. W wypadku wykrycia błędów może je skorygować.

Po poprawnym wypełnieniu wszystkich pól rekordu nowy rekord zostaje automatycznie dołączony do bazy danych. Jednocześnie automatycznie jest modyfikowany indeks autorów i wykaz słów kluczowych.

Przykład wypełnienia formularza:

AUTOR	Kowalski J.
TYTUŁ	Relacyjne bazy danych
WYDAWCA	CINTE. Prace IINTE
MIEJSCE	Warszawa
ROK	1985
WOLUMEN	No. 5

STRONY	23–27
JĘZYK	pol.
KLUCZE	bazy danych, model relacyjny
RODZAJ	utwór

SPRAWDZANIE ZAWARTOŚCI BAZY DANYCH

Funkcja ta umożliwia wyświetlenie na ekranie monitora lub wydrukowanie na drukarce wszystkich informacji o bazie danych, tj.:
- całej lub częściowej zawartości bazy danych,
- indeksu autorów,
- zestawu słów kluczowych.

Przed wydrukiem zawartości bazy danych na ekranie pojawia się informacja o liczbie rekordów w bazie. Użytkownik musi zdecydować, czy chce obejrzeć je wszystkie czy też tylko ich część; w tym ostatnim wypadku musi podać odpowiedni zakres numerów rekordów.

WYSZUKIWANIE PROSTE

Funkcja umożliwia uzyskanie odpowiedzi na pytanie proste, tj. dotyczące wartości tylko jednego atrybutu. Atrybutem tym może być:
- nazwisko autora publikacji,
- nazwa wydawcy,
- miejsce wydania,
- rok wydania,
- język publikacji,
- słowo kluczowe,
- rodzaj publikacji.

Przy wyszukiwaniu prostym zadanie użytkownika polega na wybraniu atrybutu, według którego będzie przeglądana zawartość bazy danych, oraz na podaniu jego wartości. W odpowiedzi uzyskuje się zbiór wszystkich rekordów, w których wybrany uprzednio atrybut ma podaną wartość, albo tylko samą liczbę tych rekordów.

Dla atrybutu *autor* jest możliwe uzyskanie dwóch kategorii zestawień zbiorczych: z podziałem ze względu na rodzaj publikacji lub na język publikacji.

Przykład odpowiedzi systemu w postaci zestawienia:

```
WSZYSTKIE PUBLIKACJE NAPISANE PRZEZ Kowalski J.
RAZEM ZWARTE ARTYKUŁ UTWÓR MAT.KON.
  35      5      15     3      10
SPR.NAUKOWE NORMA
     1        2
```

WYSZUKIWANIE ZŁOŻONE

Funkcja ta umożliwia uzyskanie odpowiedzi na pytania złożone, tj. wybór rekordów spełniających koniunkcję warunków występujących w pytaniach prostych. Dla pewnych atrybutów (autor, miejsce, wydawca, język, słowa kluczowe i rodzaj) dopuszczalne jest podanie do trzech wartości danego atrybutu, oddzielonych przecinkami (co jest równoważne alternatywie w warunku prostym), a dla daty wydania publikacji – przedziału czasowego (od roku – do roku).

Przykładowo, jeżeli formularz zapytaniowy dla wyszukiwania złożonego zostanie wypełniony w następujący sposób:

AUTOR	Kowalski J.
TYTUŁ	
MIEJSCE	Warszawa, Kraków
WYDAWCA	
OD ROKU	1982
DO ROKU	1984
JĘZYK	
SŁOWA KLUCZOWE	bazy danych, normalizacja w inte
RODZAJ PUBLIKACJI	artykuł

to zostaną odszukane wszystkie publikacje J. Kowalskiego na temat baz danych lub normalizacji w inte, wydane w postaci artykułów w Warszawie lub w Krakowie w latach 1982–1984. Odpowiedzią może być zbiór wszystkich rekordów lub tylko ich liczba, wyprowadzona na ekran monitora albo na drukarkę.

AKTUALIZOWANIE DANYCH

Funkcja ta umożliwia dokonywanie zmian zawartości rekordów w istniejącej już bazie danych. Możliwe jest wprowadzanie poprawek we wszystkich kolejnych rekordach lub tylko we wskazanym rekordzie.

ZMIANA NAGŁÓWKA

We wszystkich wydrukach, będących odpowiedzią systemu na zadane pytania, pojawia się standardowy nagłówek:

```
INSTYTUT INFORMACJI NAUKOWEJ,
TECHNICZNEJ I EKONOMICZNEJ
WARSZAWA
```

System umożliwia zmianę tego nagłówka i wprowadzenie własnego tekstu.

Przykład wypełnienia formularza w celu zmiany nagłówka:

```
NOWY NAGŁÓWEK
PODAJ TEKST, KTÓRY MA BYĆ W PIERWSZEJ LINII
Polska Akademia NAUK
PODAJ TEKST, KTÓRY MA BYĆ W DRUGIEJ LINII
Instytut Socjologii
PODAJ TEKST, KTÓRY MA BYĆ W TRZECIEJ LINII
Warszawa
```

Od tej chwili we wszystkich nagłówkach wydruku będzie używany tekst:

```
Polska Akademia Nauk
Instytut Socjologii
Warszawa
```

. . .

Mikrokomputerowy system bibliograficzny został praktycznie wykorzystany do założenia bibliograficznej bazy danych w Instytucie Informacji Naukowej, Technicznej i Ekonomicznej. Baza ta zawiera informacje o wszystkich publikacjach pracowników Instytutu INTE wydanych w latach 1971–1987.

System nadzorujący z bazą wiedzy

Firma Expertech wprowadziła na rynek uniwersalny zestaw sprzętowo-programowy Annie (Analogue Interface Expert) przeznaczony do budowy systemów ekspertowych nadzorujących pracę urządzeń technicznych. W skład zestawu wchodzi ogólne narzędzie budowy systemów ekspertowych Xi Plus, płyta sprzęgu analogowego oraz oprogramowanie (napisane w języku C) umożliwiające komunikację między Xi Plus i płytą. Na przykład, proponowany zestaw może na bieżąco nadzorować i oceniać warunki pracy dużego silnika wysokoprężnego Diesla. Nadzór jest prowadzony na podstawie automatycznie mierzonych wartości takich parametrów silnika, jak: temperatura, prędkość turbiny oraz prędkość przepływu gazów. Zestaw Annie zamienia numeryczne wartości pomiarowe na oceny jakościowe i na ich podstawie wyciąga wnioski o stanie pracy nadzorowanego urządzenia.

M.M.

BOGUMIŁA PAWLAK
Warszawa

MIC – metaasembler mikroprogramów

Określenie zawartości pamięci sterującej (pamięci mikroprogramu) jest jednym z najważniejszych problemów występujących podczas projektowania i uruchamiania urządzeń mikroprogramowanych. Dla każdego kroku algorytmu realizowanego przez urządzenie należy zdefiniować sygnały (ciągi zer i jedynek) sterujące poszczególnymi blokami cyfrowymi. Trudno w tym wypadku mówić o kojarzeniu takich ciągów cyfr z funkcjami realizowanymi przez blok. W mikroprogramowaniu poziomym, w którym każdy blok urządzenia ma własne pole sterujące w słowie mikrorozkazowym, liczba bitów w słowie dochodzi do 300. Bezbłędne przygotowanie zawartości pamięci dla przeciętnego mikroprogramu (200-500 mikrorozkazów) bez żadnych narzędzi wspomagających jest praktycznie niemożliwe. Jakikolwiek błąd wymaga odnalezienia i poprawienia błędnej sekwencji i cyfr pośród tysięcy zer i jedynek. Aby ułatwić tę żmudną pracę konieczne było opracowanie języków, pozwalających zapisać w sposób symboliczny zawartości pamięci sterującej, oraz translatorów przekształcających ten zapis na bitowy obraz pamięci.

Architektura urządzeń mikroprogramowanych jest bardzo różnorodna. Konstruowanie ukierunkowanego translatora mikroprogramów dla każdego urządzenia zwiększałoby znacznie czas i koszty jego produkcji. Celowe było opracowanie bardziej uniwersalnych narzędzi programowych, zapewniających możliwość definiowania zbioru mikroinstrukcji dla dowolnego układu. Narzędzia takie nazwano metaasemblerami mikroprogramów.

Metaasemblery mikroprogramów są przeznaczone do kodowania pamięci sterującej o różnych strukturach słów. Użytkownik może zdefiniować własny język do symbolicznego zapisu zawartości pamięci mikroprogramu; na podstawie tej definicji metaasembler przystosowuje swoje działanie do pracy jako żądany asembler albo tworzy na wyjściu program konkretnego asemblera. Ta różnica w działaniu jest kryterium podziału metaasemblerów [8], odpowiednio na: metaasemblery adaptacyjne (ang. *adaptive meta-assemblers*) i generatory translatorów (ang. *generative meta-assemblers*).

Jednym z najbardziej popularnych metaasemblerów jest AMDASM opracowany w firmie Advanced Micro Devices. Wiele później powstałych metaasemblerów (np. MICA, UMAS [4]) ma zbliżoną do niego składnię. Na metaasemblerze AMDASM oparto również nowy metaasembler MIC, opracowany w Instytucie Informatyki Politechniki Warszawskiej. Jego składnia jest jednak znacznie rozszerzona w stosunku do składni metaasemblera AMDASM. MIC jest częścią oprogramowania systemu uruchomieniowego mikroprogramowanych układów sterowania MIDES.

MECHANIZM PRACY I CECHY METAASEMBLERÓW

Praca typowego metaasemblera adaptacyjnego jest podzielona na trzy fazy, którym odpowiadają trzy odrębne programy.

W **fazie definicji** są generowane tabele pierwotnie definiowanych symboli asemblera, na podstawie dostarczonego zbioru definicji i deklaracji. Sformalizowany opis, wykonany za pomocą języka metaasemblera, obejmuje zdefiniowanie stałych i mikroinstrukcji – określenie ich długości oraz formatów. Dyrektywy są stałą częścią języka metaasemblera i nie są definiowane. W fazie tej jest wypełniana tabela, łącząca nazwy formatów i stałych, nadane przez użytkownika z odpowiadającymi im wzorcami bitowymi.

Następnym etapem jest **translacja mikroprogramu** napisanego w języku zdefiniowanym w fazie pierwszej. Program symboliczny, w którym są dopuszczalne etykiety i ustawianie licznika adresów, jest odczytywany i przetwarzany na bitowy obraz pamięci sterującej. W fazie tej jest drukowany raport pracy i tablica odwołań.

Wynikiem fazy translacji jest obraz pamięci sterującej w postaci jednego bloku o wymiarach równych długości mikroinstrukcji i maksymalnemu adresowi określonemu w mikroprogramie. Rzeczywista struktura pamięci mikroprogramowanego układu sterowania jest zwykle inna. Dopasowanie struktury logicznej do struktury rzeczywistej jest celem trzeciej fazy metaasemblera – **fazy posttranslacyjnej**. W fazie tej pamięć mikroprogramu jest dzielona na segmenty odpowiadające poszczególnym układom pamięci stałej. Zawartość pamięci może być wysyłana bezpośrednio do programatora pamięci stałych lub wyprowadzana na nośnik zewnętrzny w formacie właściwym dla programatora.

Obecnie wiele metaasemblerów mikroprogramów działa zgodnie z opisanym mechanizmem. Różnią się one między sobą możliwościami, przeznaczeniem oraz szczegółowymi sposobami zapisu. Nie warto robić przeglądu metaasemblerów (można go znaleźć np. w [8]), ale analiza ich właściwości pozwala sformułować zbiór cech, jakie powinien mieć dobry metaasembler mikroprogramów.

Program metaasemblera powinien być napisany w języku wysokiego poziomu w sposób niezależny od systemu, aby zapewnić jego **przenośność**. Bardzo przydatny do tego celu jest język C, który oprócz wszystkich zalet języków wysokiego poziomu, zawiera mechanizmy umożliwiające optymalizację generowanego kodu (pola, instrukcje warunkowe, zmienne rejestrowe) oraz dopuszcza operacje typowe dla języków asemblerowych, np. operacje na bitach. W języku C jest napisany najnowszy metaasembler firmy Advanced Micro Devices – M29 [3]. Wcześniejsze przenośne metaasemblery były pisane w Fortranie (np. AMDASM, DEFASM) i Pascalu (np. MICRO8).

Metaasembler służy do efektywnej i prostej realizacji nowego asemblera. **Język metaasemblera powinien być elastyczny i łatwy do użycia**, aby opracowanie zbioru definicji było o wiele prostsze niż napisanie własnego ukierunkowanego asemblera.

Dobry **metaasembler dopuszcza realizację asemblerów dla języków o wielu możliwościach**, jak np. modularyzacja mikroprogramu (również na podprogramy), obliczanie wyrażeń, asemblacja warunkowa, mechanizmy makrogeneracji. Większość metaasemblerów dopuszcza jedynie wartościowanie wyrażeń. Wyjątkiem jest M29, który ma realizować wszystkie wymienione mechanizmy [3].

Nową cechą wśród metaasemblerów mikroprogramów jest **tworzenie przez metaasembler kodu relokowalnego**. Umożliwia to niezależną translację segmentów i tworzenie bibliotek mikroprogramów. Wiąże się to również z koniecznością powstania programu zarządzającego biblioteką (np. program M29LIB w systemie M29) oraz konsolidatora mikrokodu (np. M29LINK), który ma przypisywać adresy poszczególnym modułom translacji, realizować powiązania między nimi i przeszukiwać biblioteki.

Dużym ułatwieniem podczas kodowania pamięci mikroprogramu jest sygnalizacja elementarnych błędów. Zapewnić to mogą jedynie metaasemblery wyposażone **w mechanizmy kontroli mikroprogramu**. Często stosowanym sposobem jest ustalenie, dla każdego pola formatu, listy dozwolonych mnemoników i odpowiadającym im wartości. Mechanizm ten ma różne nazwy i składnię zapisu, zależnie od metaasemblera, np. ASSIGN w metaasemblerze DEFASM [5], MICROPS w XMAS, specyfikacja wartości w M29. Inna funkcja kontrolna (IMPLY w DEFASM i XMAS) umożliwia określenie i sprawdzenie zależności między wartościami umieszczonymi w róż-

nych polach mikroinstrukcji. Niestety, nie we wszystkich metaasemblerach wspomniane mechanizmy są realizowane (nie ma ich np. w metaasemblerach AMDASM i RAYASM). Większość metaasemblerów ogranicza się jedynie do sygnalizowania podstawowych błędów, jak np. użycie niezdefiniowanego symbolu lub podwójna definicja etykiety.

Wśród istniejących metaasemblerów nie można znaleźć takiego, który miałby wszystkie opisane cechy. Jak wynika z zapowiedzi, M29 może być pierwszym spełniającym wszystkie te wymagania, ale jego realizacja nie została jeszcze potwierdzona w dostępnych autorce publikacjach.

METAASEMBLER AMDASM

Metaasembler AMDASM jest typowym metaasemblerem adaptacyjnym. Poszczególne fazy jego pracy (generowanie tabeli symboli, translacja, przetwarzanie obrazu pamięci) mogą być wykonywane wielokrotnie i niezależnie. Umożliwia to użytkownikowi jednorazowe zdefiniowanie własnego języka, którym posługuje się zapisując różne mikroprogramy.

W pliku definicji jako pierwsza jest definiowana długość słowa mikroinstrukcji. Słowo może mieć długość od 1 do 128 bitów; na przykład dyrektywa

WORD 29

definiuje słowo o długości 29 bitów.
Dalej następują definicje stałych i formatów mikroinstrukcji. Definicja stałej przypisuje nazwie symbolicznej odpowiedni wzorzec bitowy. Definicja formatu określa zawartość mikroinstrukcji jako zbiór pól o ustalonej długości i stałej lub zmiennej zawartości. Ciągi pól powtarzające się w wielu formatach mogą być zadeklarowane jako podformaty, które będą wykorzystywane podczas definiowania formatów.
Przykładowo:

RAMF: SUB B # 01,1X

określa podformat o długości 3 bitów, składający się z 2-bitowego pola stałego o wartości 01 (dwójkowo) i 1-bitowego pola nieistotnego.

AMDASM umożliwia definiowanie stałych lub pól stałych za pomocą liczb zapisanych w różnych kodach: dwójkowym, ósemkowym, dziesiętnym lub szesnastkowym; na przykład, fragment:

OR: EQU Q # 3

jest definicją stałej *OR* o długości 3 bitów i wzorcu bitowym 011, zapisaną w kodzie ósemkowym.

Dopasowanie wzorca bitowego liczb do wymagań sprzętowych ułatwiają modfikatory, za pomocą których można negować wartość stałej (stosowane, gdy występują bufory negujące) oraz rozszerzać lub skracać wzorzec bitowy stałej do żądanej liczby bitów.

Format jest definiowany jako ciąg pól stałych, zmiennych i nieistotnych, na przykład:

load: DEF 24X,B # 0,4V%:

Format *load* składa się z pola nieistotnego o długości 24 bitów, 1-bitowego pola stałego o wartości 0 i 4-bitowego pola zmiennego. Modyfikatory „%" i „:" przypisane polu zmiennemu powodują, że długość parametru aktualnego dla tego pola zostanie dopasowana (skrócona lub rozszerzona) do długości pola.

Pola zmienne są to części mikroinstrukcji, których wzorce bitowe zmieniają się podczas każdego użycia formatu. Zawartość pola zmiennego jest określana dopiero w fazie transalacji przez podanie parametru aktualnego. Przykładem pola zdefiniowanego jako pole zmienne sa bitv określające argument operacji, np. *3VQ #* definiuje pole zmienne o długości 3 bitów; jako domniemany typ parametru aktualnego dla tego pola określono typ ósemkowy. W definicji pola zmiennego można określić jego wartość domniemaną – wzorzec bitowy, który należy przypisać polu, jeśli w mikroprogramie parametr aktualny dla tego pola zostanie pominięty; przykładowo *1VB#0* definiuje 1-bitowe pole zmienne o wartości domniemanej 0. Jeśli jako wartość domniemana zostanie wybrana najczęściej powtarzająca się wartość pola, to pozwoli to skrócić zapis mikroprogramu zapewniając jednocześnie odpowiednie sterowanie układem.

Oprócz pól stałych i zmiennych występujących w formatach większości metaasemblerów, w metaasemblerze AMDASM wprowadzono dodatkowo pola nieistotne, np. *5X* – oznacza pole nieistotne o długości 5 bitów. Pola nieistotne wyznaczają te bity słowa mikroinstrukcji, które nie są używane w konkretnym formacie, np. pole adresowe w mikrooperacji kontynuacji. Wartości bitów nieistotnych mogą zostać ustalone w fazie translacji przez nakładanie formatów – kiedy pola nieistotne jednego formatu pokrywają się z polami stałymi lub zmiennymi innego formatu. Bity nieistotne, które nie zostały zdefiniowane w fazie translacji, będą określone przez użytkownika dopiero podczas fazy posttranslacyjnej. Najczęściej jest im nadawana wartość bitów pamięci stałej przed zaprogramowaniem. Powyższe właściwości pól nieistotnych są bardzo przydatne podczas uruchamiania, gdy optymalizuje się długość słowa mikroinstrukcji przez łączenie pól.

W fazie translacji plik mikroprogramu zawiera mikroinstrukcje, które są symbolicznym zapisem zawartości słów pamięci sterującej. Są one tworzone przez odwołanie się do nazwy formatu zdefiniowanego w fazie definicji lub bezpośrednio w pliku mikroprogramu przez użycie formatu swobodnego. Słowa dwójkowe powstałe z formatów mikroinstrukcji i odpowiadających im list parametrów aktualnych mogą być sumowane logicznie bit po bicie w celu otrzymania kompletnego słowa mikroinstrukcji. Sumowanie takie nazywa się nakładaniem formatów. Nakładanie formatów jest dozwolone tylko wówczas, gdy bity istotne jednego formatu nakładają się z bitami nieistotnymi pozostałych nakładanych mikroinstrukcji.
Przykładowo:

M2:F(=AD.(D,0)&Q:=F&exit.(Z=1).M2

mikroinstrukcja o etykiecie *M2* została otrzymana przez nakładanie trzech formatów o nazwach: *F(=, Q:=F* i *exit* wraz z ich parametrami aktualnymi; dla formatu *F(=* parametrami aktualnymi są stałe *AD* i *.(D,O).*

Słowo dwójkowe jest zapisywane do pamięci pod adres równy wartości wskaźnika wprowadzania. Użytkownik musi zapisywać mikroinstrukcje wykonawcze w pliku mikroprogramu w takiej kolejności, w jakiej odpowiadające im słowa dwójkowe mają być umieszczone w pamięci sterującej. Kolejność tę można zmienić przez użycie dyrektyw zmiany wartości wskaźnika wprowadzania. Z rozmieszczaniem mikroprogramu w pamięci i adresowaniem symbolicznym jest związana kontrola stronicowania pamięci, sygnalizowana specjalnym modyfikatorem.

Oprócz tej funkcji AMDASM nie zawiera innych mechanizmów kontroli poprawności mikroprogramu. Jest to najistotniejsza wada tego metaasemblera, uniemożliwiająca natychmiastowe wykrywanie elementarnych błędów. Uciążliwe są również zbyt ostre ograniczenia na długość pól i niekonsekwencje w składni.

W metaasemblerze MIC usunięto powyższe wady przez wprowadzenie dodatkowych konstrukcji i zwiększenie wartości ograniczających.

METAASEMBLER MIC

Metaasembler MIC jest zgodny z metaasemblerem AMDASM, tzn. każdy mikroprogram zgodny ze składnią AMDASM będzie poprawnie przetworzony przez MIC. MIC jest metaasemblerem adaptacyjnym o pracy podzielonej na trzy fazy, którym odpowiadają trzy odrębne programy. Programy te zostały napisane w języku C i uruchomione pod kontrolą systemu operacyjnego MS-DOS na komputerze IBM PC. MIC jest całkowicie niezależny od konfiguracji sprzętu a jego realizacja w języku wysokiego poziomu zapewnia działanie również w innych systemach (po dokonaniu drobnych zmian). MIC

ma bardzo wygodny sposób komunikowania się z użytkownikiem i daje duże możliwości oddziaływania na postać powstających raportów pracy.

Składnia języka metaasemblera MIC jest znacznie rozszerzona w stosunku do składni AMDASM oraz do pierwszej implementacji AMDASM w Instytucie Informatyki PW – UMAS [4]. Wprowadzono dwa mechanizmy kontroli mikroprogramu. Pierwszy z nich jest wzorowany na analogicznym mechanizmie z metaasmblera MICA. W fazie definicji wprowadzono dodatkową konstrukcję SYMGRP (SYMBOL GROUP) umożliwiającą łączenie w grupę nazw (symboli) wcześniej zdefiniowanych stałych, przykładem może być grupa funkcji jednostki arytmetyczno-logicznej, ALU:

FUN: SYMGRP AD,AND,OR

Podczas deklarowania formatu mikroinstrukcji każdemu polu zmiennemu można przypisać określoną grupę symboli, przykładowo, w definicji *3V(FUN)* polu zmiennemu została przypisana grupa symboli *FUN*. Parametr aktualny dla tego pola musi być nazwą stałej należącą do grupy funkcji *FUN*.

Kontrola w fazie translacji polega na sprawdzeniu, czy parametr aktualny (symbol) należy do grupy symboli określonej dla danego pola zmiennego. Wprowadzona konstrukcja jest jednym z najsilniejszych mechanizmów wykrywania i tym samym unikania błędów. Większość metaasemblerów zawiera podobnie działające funkcje, a jej zrealizowanie w metaasemblerze MIC znacznie ułatwia poprawne kodowanie zawartości pamięci.

Drugim mechanizmem kontroli jest wprowadzenie dodatkowego stanu dla pól zaznaczonych w definicji formatu jako nieistotne. Chociaż wartość takich pól nie jest jawnie precyzowana (np. w celu późniejszego nakładania formatów), to jednak niekiedy nie powinna być przypadkowa. Celowe jest więc rozróżnienie dwóch rodzajów pól nieistotnych: pól dowolnych oznaczonych przez *X* (tak jak w AMDASM) oraz pól określanych, dla których wprowadzono dodatkowe oznaczenie *Y*. W fazie translacji, po nałożeniu formatów i skomplikowaniu całego słowa mikroinstrukcji następuje sprawdzenie, czy wszystkie pola zaznaczone jako *Y* zostały określone.

Metaasembler MIC w fazie translacji pozwala zdefiniować tzw. format bitów nieistotnych. Konstrukcja ta umożliwia nadanie ustalonej wartości bitom słowa, które pozostaną nieokreślone po nałożeniu formatów. Zbędne staje się nakładanie w każdej mikroinstrukcji jeszcze jednego formatu definiującego właśnie te bity, gdyż zostanie to wykonane automatycznie. Przykładowa definicja formatu bitów nieistotnych:

XF 9X,B#0, 14X,1,4X

określa bit 4 i 19 słowa pamięci, odpowiednio na wartości 1 i 0.

Maksymalna długość słowa mikroinstrukcji w metaasemblerze MIC wynosi 256 bitów, co zapewnia zgodność z systemem uruchomieniowym MIDES.

W metaasemblerze AMDASM przyjęto stałą maksymalną długość pola stałego i zmiennego równą 16 bitów, co znacznie utrudniało definiowanie formatów. Pole dłuższe należało sztucznie dzielić na mniejsze fragmenty, tracąc przez to zwięzłość i czytelność zapisu. MIC umożliwia użytkownikowi zdefiniowanie maksymalnej długości pola i stałej przez wprowadzenie dodatkowej dyrektywy *MAXLEN*. Brak tej dyrektywy powoduje przyjęcie przez domniemanie wartości 16.

Wprowadzono również wiele rozszerzeń usuwających uciążliwe ograniczenia i wymagania metaasemblera AMDASM, np. dozwolono nakładanie bitów o tych samych wartościach, umożliwiono zapis liczby określającej długość stałej lub pola w każdym kodzie akceptowanym przez MIC.

Największe różnice w stosunku do pierwowzoru występują w trzeciej fazie metaasemblera – fazie posttranslacyjnej. Głównym zadaniem tego etapu jest wyznaczenie zawartości kolejnych układów pamięci mikroprogramowanego układu sterowania na podstawie zawartości pamięci sterującej wygenerowanej w fazie translacji. Aby umożliwić połączenie różnych fragmentów pamięci, zapisanych w postaci oddzielnie translowanych mikroprogramów, dopuszcza się określenie wielu plików zawierających obraz pamięci. Metaasembler wczytuje treść tych plików i jeśli powstały w ten sposób obraz pamięci nie jest uporządkowany, to sortuje go w kolejności rosnących adresów. Takie rozwiązanie pozwala, chociaż w sposób bardzo ograniczony, dzielić mikroprogram na moduły. Użytkownik musi jednak sam zapewnić połączenia między nimi.

Faza trzecia odpowiada programowi konwersacyjnemu, przez który użytkownik określa organizację układów pamięci oraz wpływa na rodzaj i wielkość informacji przekazywanej do programatora pamięci. Po skompletowaniu i uporządkowaniu obrazu pamięci, ekran monitora jest dzielony na dwa okienka. W dolnym metaasembler wyświetla pytania dotyczące organizacji pamięci. Odpowiadając na nie użytkownik określa strukturę układów pamięci mikroprogramowanego układu sterowania. Schemat zadanej struktury w postaci prostokątów jest wyświetlany w górnym okienku. W każdym „prostokącie pamięci" jest podany jej numer, przez który użytkownik może odwoływać się do konkretnego układu. Rozmiary bloku układów pamięci powinny być równe lub większe niż rozmiary logicznego obrazu pamięci. W tym drugim wypadku nadmiarowe bity lub słowa są wypełniane bitami nieistotnymi; wartość bitów nieistotnych określa użytkownik. Użytkownik podaje również nazwę pliku, w którym metaasembler umieści zawartość wskazanych układów pamięci w formacie właściwym dla programatora pamięci stałych. Istnieje możliwość negowania adresów oraz utworzenia pliku zawierającego obraz całej pamięci w postaci wymaganej przez symulator pamięci stałych w systemie uruchomieniowym MIDES.

PRZYKŁAD DZIAŁANIA METAASEMBLERA

Działanie metaasemblera zostanie zilustrowane na przykładzie określenia zawartości pamięci sterującej układu mnożącego metodą podstawową dwie liczby zapisane w naturalnym kodzie dwójkowym (NKB). Mikroprogram opisujący algorytm działania układu oraz projekt części wykonawczej i sterującej układu przedstawiono w [7].

Na wydruku 1 przedstawiono opis zawartości pliku definicji, w którym określono formaty i stałe wykorzystane później do zakodowania sygnałów sterujących. W pliku tym zdefiniowano jedynie symbole niezbędne do zapisania zawartości pamięci dla tego konkretnego algorytmu.

```
;Plik definicji
WORD 29

;Argumenty mikrooperacji w Am2901
.(D,0):                 EQU     0#7
.(0,RAM[B]):            EQU     0#3
.(RAM[A],RAM[B]):       EQU     0#1

ARG:    SYMGRP  .(D,0)..(0,RAM[B])..(RAM[A],RAM[B])

;Funkcje Am2901
AD:     EQU     0#0
AND:    EQU     0#4
OR:     EQU     0#3

FUN:    SYMGRP  AD,AND,OR

;Mikrooperacje w RAM,Q,UP1,UP2 w Am2901
QF:             EQU     0#0
RAMF:           SUB     B#01,1X
SHR.RAM.Q:      EQU     0#4

;Warunki rozejscia
.(Z=1).:        EQU     B#010011
.(Z=0).:        EQU     B#011001
.(Q=1).:        EQU     B#10
.(Q0=1).:       EQU     B#000011
.(L'=[0]).:     EQU     B#11

COND:   SYMGRP  .(Z=1)...(Z=0)...(Q=1)...(Q0=1)...(L'=[0]).

;Sterowanie Am2901
A<=:    DEF     14X,4VH#,11X
B<=:    DEF     10X,4VH#,15X
F<=:    DEF     3X,3V(FUN),3V(ARG),1VB#0,19X
Q:=F:   DEF     QF,26X
RAM(B):=F:                      DEF     RAMF,26X
RAM(B):=SHR.F..Q:=SHR.Q:        DEF     SHR.RAM.Q,26X

;Sterowanie Am29811
exit:                   DEF     18X,6V,1X,4V%:
L:=7.exit.next:         DEF     20X,H#C,5X
load:                   DEF     24X,B#0,4V%:
L:=DEC(L).exit:         DEF     18X,2V(COND),H#9,1B#1,4V%:
R.exit:                 DEF     18X,2V(COND),H#7,B#1,4V%:

END
```

Wydruk 1. Plik definicji

Wydruk 2 jest raportem fazy translacji. Raport ten zawiera kolejne mikroinstrukcje, zapisane za pomocą symboli określonych w pierwszej fazie, wraz z postaciami dwójkowymi utworzonymi przez *MIC2*. Ostatnim elementem wydruku jest obraz pamięci sterującej w postaci jednego bloku.

```
1:  ;    Mikroprogram
2:
3:  ;format bitow nieistotnych
4:  XF  9X,0.14X,B#1,4X
Format of don't care bits:
          XXXXX XXXXOXXX XXXXXXXX XXX1XXXX


6:  M1: F<= AD .(D,0) & B<= 0 & RAM(B):=F &
7:          exit .(Z=0). M1 ; .(Z=1).M2
  0000:     01X00 01110000 0XXXX011 00110000
8:  M2: F<= AD .(D,0) & D:=F & exit .(Z=1). M2 ; .(Z=0).M3
  0001:     00000 01110XXX XXXXX010 01110001
9:  M3: B<= 1 & F<= AND .(0,RAM(B)) & RAM(B):=F & load M7 &
10:         L:=7.exit.next
  0002:     01X10 00110000 1XXXXXX1 10000111
11: M4:       B<= 1 & F<= OR .(0,RAM(B)) & RAM(B):=SHR.F..0:=SHR.0 &
12:         exit .(D0=1). M6 ; .(D0=0).M5
  0003:     10001 10110000 1XXXX000 01110110
13: M5:       B<= 1 & F<= AD .(0,RAM(B)) & RAM(B):=F &
14:         L:=DEC(L).exit .(L!=[0]). M4 ; .(L=[0]).M7bis
  0004:     01X00 00110000 1XXXX111 00110011
15: M7bis:    B<= 1 & F<= OR .(0,RAM(B)) & RAM(B):=F &
16:         R.exit .(D=1). M1 ; .(D=0).M7
  0005:     01X01 10110000 1XXXX100 11110000
17: M6:       A<= 0 & B<= 1 & F<= AD .(RAM(A),RAM(B)) & RAM(B):=F &
18:         L:=DEC(L).exit .(L!=[0]). M4 ; .(L=[0]).M7
  0006:     01X00 00010000 10000111 00110011
19: M7:       B<= 1 & F<= OR .(0,RAM(B)) & RAM(B):=F &
20:         R.exit .(D=1). M1 ; .(D=0).M7
  0007:     01X01 10110000 1XXXX100 11110000
21: END


  0000:     01X00 01110000 0XXXX011 00110000
  0001:     00000 01110XXX XXXXX010 01110001
  0002:     01X10 00110000 1XXXXXX1 10000111
  0003:     10001 10110000 1XXXX000 01110110
  0004:     01X00 00110000 1XXXX111 00110011
  0005:     01X01 10110000 1XXXX100 11110000
  0006:     01X00 00010000 10000111 00110011
  0007:     01X01 10110000 1XXXX100 11110000
```

Wydruk 2. Raport fazy translacji

Korzystając z metaasemblera użytkownik opracowuje zwykle własny język, który wykorzystuje do kodowania różnych algorytmów. W pliku definicji jest definiowanych wiele formatów słów sterujących oraz stałe symbolizujące możliwe wartości pól tych formatów. Dla takiego pełnego zbioru symboli jest wykonywana jednokrotnie faza definicji, a utworzone przez nią tabele symboli są wielokrotnie wykorzystywane podczas translacji różnych mikroprogramów. Korzyści z zastosowania metaasemblera są najwyraźniej widoczne przy wielokrotnym zapisie dużych pamięci sterujących za pomocą raz zdefiniowanych symboli. Przytoczenie odpowiedniego przykładu przekracza niestety ramy niniejszego artykułu.

. . .

Metaasembler MIC powstał jako praca dyplomowa [6] w Instytucie Informatyki Politechniki Warszawskiej. Obecnie jest on wykorzystywany przez pracowników i studentów Instytutu, m. in. w laboratorium projektowania układów mikroprogramowanych. Dotychczasowe wykorzystanie metaasemblera dowiodło jego użyteczności. Doświadczenia użytkowników są na bieżąco analizowane, a wynikające z nich celowe modyfikacje są uwzględnione w kolejnych wersjach metaasemblera MIC.

LITERATURA

[1] AMDASM Reference Manual. The Am2900 Family Data Book. Advanced Micro Devices, 1978

[2] Dasgupta S.: The Design and Description of Computer Architectures. John Wiley, New York, 1984

[3] Eager M.J.: M29 – Advanced Retargetable Microcode Assembler. Proc. 16th Ann. Microprogramming Workshop MICRO-16, ACM SIGMICRO NEWSLETTER, Vol. 14, No.4, 1983

[4] Kowalski R.: Oprogramowanie systemu wspomagającego uruchamianie układów mikroprogramowanych. Praca magisterska Instytutu Informatyki, Politechnika Warszawska, 1983

[5] Mezzalama M., Prinetto P., Romani S.: DEFASM: a microprogram meta-assembler with semantic capability. IEEE Proc., Vol. 128, No. 4, July 1981

[6] Pawlak B.: Metaasembler MIC jako składnik oprogramowania systemu uruchomieniowego mikroprogramowanych układów sterowania. Praca magisterska Instytutu Informatyki, Politechnika Warszawska, 1987

[7] Pawłowski M.: Mikroprogramowanie. Informatyka nr 4 i 5, 1987

[8] Skordalakis E.: Meta-assemblers. IEEE Micro, Vol.3, No.2, April 1983.

ADAM TUCHOLSKI
Instytut Łączności
Zakład Teleinformatyki
Warszawa

NETBIOS – zasada działania i sposób użytkowania (2)

Dokończenie artykułu nt. NETBIOS-a zawiera przykłady użycia NETBIOS-a poparte dużym fragmentem programu.

PRZYKŁADY

Poniżej przedstawiono kilka przykładów przesyłania informacji w sieci.

Inicjowanie dołączenia użytkownika do sieci (konieczne, aby jakiekolwiek przesłanie mogło być zrealizowane)

- Wykonujemy polecenie *RESET* (nie jest to konieczne, gdy przyjmujemy parametry domyślne);
- Tworzymy blok *ADD_NAME* z własną nazwą. Pola nie wykorzystane powinny być wyzerowane;
- Wywołujemy NETBIOS przy użyciu rozkazu *INT 5CH*.

Wysyłanie połączeniowe danych

- Tworzymy blok *CALL_NCB* wypełniając odpowiednie pola (zerując pola nie wykorzystane);
- Wywołujemy NETBIOS przy użyciu rozkazu *INT 5CH;*
- Sprawdzamy kod powrotny w rejestrze *AL* po zakończeniu polecenia *CALL*;

```
;*****************************************************************
;Niniejszy fragment programu umożliwia nadanie/odebranie danych *
;w trybie połączeniowym lub bezpołączeniowym                    *
;*****************************************************************
;
;======= STRUKTURY
;------- NETWORK CONTROL BLOCK
NCB             STRUC
COMMAND         DB    00H
RETCODE         DB    00H
LSN             DB    00H
NUM             DB    00H
BUFFER@         DD    00000000H
LENGTH          DW    0000H
CALLNAME        DB    16 DUP(0)
NAME            DB    16 DUP(0)
RTO             DB    00H
STO             DB    00H
POST@           DD    00000000H
LANA_NUM        DB    00H
CMD_CPLT        DB    00H
RESERVE         DB    14 DUP(0)
NCB             ENDS
;
;======= SEGMENT STOSU
STACK_SEG       SEGMENT       'STACK'
                DB    120 DUP(0)    ; 120-bajtów zarezerwowanych na stosie
STACK_BASE      LABEL WORD
STACK_SEG       ENDS
;
;======= SEGMENT DODATKOWY
EXTRA_SEG       SEGMENT       'EXTRA'
;------- NETWORK CONTROL BLOCKS
ADD_NAME        NCB   <30> ; tryb synch.
CALL_NCB        NCB   <10,.....,40,40> ; tryb synch., przeterminowanie 20s
LISTEN          NCB   <91,.....,40,40> ; tryb asynch., przeterminowanie 20s
HANG_UP         NCB   <12> ; tryb synch.
SEND_DGD        NCB   <22> ; tryb synch.
RECE_DGD        NCB   <23> ; tryb synch.
EXTRA_SEG       ENDS
;
;======= SEGMENT DANYCH
;------- BUFORY
DATA_SEG        SEGMENT       'DATA'
NAME_1          DB    'KRISTINE'    ; nazwa użytkownika
NAME_2          DB    'ALICIA'      ; nazwa użytkownika wywoływanego
SEND            DB    'INFORMATYKA' ; dane nadawane
RECEIVE         DB    512 DUP(0)    ; rezerwacja pamięci dla odbioru
DATA_SEG        ENDS
;
;======= SEGMENT PROGRAMU
CODE_SEG        SEGMENT       'CODE'
ASSUME          CS:CODE_SEG, SS:STACK_SEG, DS:DATA_SEG, ES:EXTRA_SEG
START:          MOV   AX,DATA_SEG           ;ustawianie segmentu danych
                MOV   DS,AX
                MOV   AX,EXTRA_SEG          ; ustawianie segmentu dodatkowego
                MOV   ES,AX
                MOV   AX,STACK_SEG          ; ustawianie segmentu stosu
                MOV   SS,AX
                MOV   SP,OFFSET STACK_BASE ; ustawienie wskaznika stosu
;------- NADANIE NAZWY
                MOV   BX,OFFSET ADD_NAME
                MOV   CX,SIZE NAME_1
                MOV   SI,OFFSET NAME_1      ; nazwa użytkownika
                MOV   DI,OFFSET ADD_NAME.NAME
        REP     MOVSB
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_1
                JMP   ERROR
;
;======= PRZESŁANIE POŁĄCZENIOWE
;------- ZESTAWIANIE POŁĄCZENIA
DALEJ_1:        MOV   BX,OFFSET CALL_NCB
                MOV   CX,SIZE NAME_1
                MOV   SI,OFFSET NAME_1      ; nazwa użytkownika
                MOV   DI,OFFSET CALL_NCB.NAME
        REP     MOVSB
                MOV   CX,SIZE NAME_2
                MOV   SI,OFFSET NAME_2      ; nazwa użytkownika wywoływanego
                MOV   DI,OFFSET CALL_NCB.CALLNAME
        REP     MOVSB
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_2
                JMP   ERROR
;------- PRZESŁANIE DANYCH
DALEJ_2:        MOV   BX,OFFSET CALL_NCB
                MOV   ES:[BX].COMMAND,14H  ; kod polecenia SEND, tryb synch.
                MOV   AX,OFFSET SEND       ; adres bufora
                MOV   ES:WORD PTR CALL_NCB.BUFFER@,AX
                MOV   WORD PTR ES:[BX].BUFFER@+2,DS
                MOV   ES:CALL_NCB.LENGTH,SIZE SEND   ; długość bufora
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_3
                JMP   ERROR
;------- ROZŁĄCZANIE POŁĄCZENIA
DALEJ_3:        MOV   BX,OFFSET HANG_UP
                MOV   AL,ES:LISTEN.LSN      ; numer sesji
                MOV   ES:[BX].LSN,AL
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_4
                JMP   ERROR
DALEJ_4:
;                *   dalsza część programu
;                *
;                *
;
;======= ODBIÓR POŁĄCZENIOWY
;------- ZESTAWIANIE POŁĄCZENIA
                MOV   BX,OFFSET LISTEN
                MOV   AX,OFFSET LISTEN_INT
                MOV   WORD PTR ES:[BX].POST@,AX    ; adres obsługi przerwania
                MOV   WORD PTR ES:[BX].POST@+2,CS  ; dla trybu asynch.
                MOV   CX,SIZE NAME_1
                MOV   SI,OFFSET NAME_1      ; nazwa użytkownika
                MOV   DI,OFFSET LISTEN.NAME
        REP     MOVSB
                MOV   CX,SIZE NAME_2
                MOV   SI,OFFSET NAME_2      ; nazwa użytkownika wywoływanego
                MOV   DI,OFFSET LISTEN.CALLNAME
        REP     MOVSB
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_5
                JMP   ERROR
;------- ODBIÓR DANYCH
DALEJ_5:        MOV   BX,OFFSET LISTEN
                MOV   ES:[BX].COMMAND,15H ; kod polecenia RECEIVE,tryb synch.
                MOV   AX,OFFSET RECEIVE          ; adres bufora
                MOV   WORD PTR ES:[BX].BUFFER@,AX
                MOV   WORD PTR ES:[BX].BUFFER@+2,DS
                MOV   LISTEN.LENGTH,SIZE RECEIVE  ; długość bufora
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_6
                JMP   ERROR
;------- ROZŁĄCZANIE POŁĄCZENIA
DALEJ_6:        MOV   BX,OFFSET HANG_UP
                MOV   AL,ES:LISTEN.LSN          ; numer sesji
                MOV   ES:[BX].LSN,AL
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_7
                JMP   ERROR
DALEJ_7:
;               * dalsza część programu
;               *
;               *
LISTEN_INT      LABEL DWORD
                OR    AL,AL
;               *   obsługa przerwania (zakończenie
;               *   realizacji polecenia LISTEN)
;               *
                IRET
```

```
;******* PRZESLANIE DATAGRAMOW
;------- NADANIE DATAGRAMU
                MOV   BX,OFFSET SEND_BRD
                MOV   AX,OFFSET SEND              ; adres bufora
                MOV   WORD PTR ES:[BX].BUFFER@,AX
                MOV   WORD PTR ES:[BX].BUFFER@+2,DS
                MOV   ES:[BX].LENGTH,SIZE SEND    ;długość bufora
                MOV   AL,ES:ADD_NAME.NUM          ; numer nazwy użytkownika
                MOV   ES:[BX].NUM,AL
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_8
                JMP   ERROR
;------- ODBIOR DATAGRAMU
DALEJ_8:        MOV   BX,OFFSET RECE_BRD
                MOV   AX,OFFSET RECEIVE           ; adres bufora
                MOV   WORD PTR ES:[BX].BUFFER@,AX
                MOV   WORD PTR ES:[BX].BUFFER@+2,DS
                MOV   ES:[BX].LENGTH,SIZE RECEIVE ; długość bufora
                MOV   AL,ES:ADD_NAME.NUM          ; numer nazwy użytkownika
                MOV   ES:[BX].NUM,AL
                INT   5CH
                OR    AL,AL
                JZ    DALEJ_9
                JMP   ERROR
DALEJ_9:
;               *   dalsza część programu
;               *
;               *
ERROR           LABEL NEAR
;               *   obsługa błędów
;               *
;               *
;
CODE_SEG        ENDS
                END   START
```

Przykład programu umożliwiającego nadawanie i odbieranie danych w trybie połączeniowym lub bezpołączeniowym

- Używamy poprzedniego bloku *CALL_NCB* zmieniając kod polecenia na *SEND* w odpowiednim polu bloku *CALL_NCB*;
- Określamy w bloku *CALL_NCB* bufor informacji nadawanej;
- Generujemy rozkaz *INT 5CH*;
- Kiedy przesłanie jest zakończone, używamy polecenia *HANG UP*.

Odbieranie połączeniowe danych w trybie asynchronicznym

- Tworzymy blok *LISTEN* zerując pola nie wykorzystane i wybierając tryb asynchroniczny;
- Jeśli następuje wywołanie, to wykonujemy skok do adresu obsługi przerwania (pole *NCB_POST*ⓐ);
- Po nawiązaniu połączenia należy użyć polecenia *RECEIVE* (tworząc odpowiedni blok *NCB* i generując rozkaz *INT 5CH*) dla odbioru danych od użytkownika żądającego połączenia;
- Można użyć również polecenia *RECEIVE ANY* (tworząc odpowiedni blok *NCB* i generując rozkaz *INT 5CH*) dla odbioru danych od dowolnego użytkownika, który ma z naszą nazwą zestawione połączenie.

Wysyłanie danych typu rozgłaszanego

- Korzystamy z polecenia *SEND BROADCAST DATAGRAM* (tworząc odpowiedni blok *NCB* i generując rozkaz *INT 5CH*).

Odbieranie danych typu rozgłaszanego

- Korzystamy z polecenia *RECEIVE BROADCAST DATAGRAM* (tworząc odpowiedni blok *NCB* i generując rozkaz *INT 5CH*);
- Polecenie odbioru *RECEIVE BROADCAST DATAGRAM* musi być wydane przed użyciem polecenia nadawania, w przeciwnym wypadku dane będą stracone.

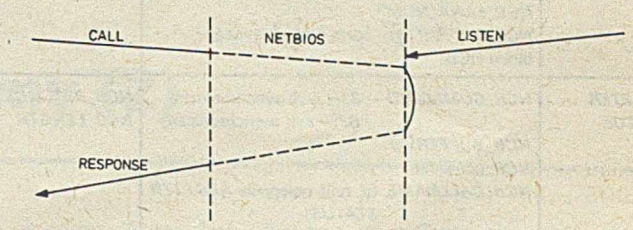
Wydruk zawiera duży przykład programu umożliwiającego nadawanie i odbieranie danych w trybie połączeniowym lub bezpołączeniowym.

UWAGI KOŃCOWE

Kluczową sprawą we wszelkich rozwiązaniach sieciowych jest gospodarka buforami. Aby połączenie mogło być nawiązane, muszą być zarezerwowane bufory (tzw. kredyt połączeniowy). W wyższych warstwach, w celu zapewnienia odpowiednich buforów do efektywnej transmisji stosuje się sterowanie przepływem. Dla przesyłania bezpołączeniowego (datagramów), korzysta się z buforów (do odbioru) uprzednio zarezerwowanych przez nadawcę (polecenia odbioru datagramów mają zasięg lokalny).

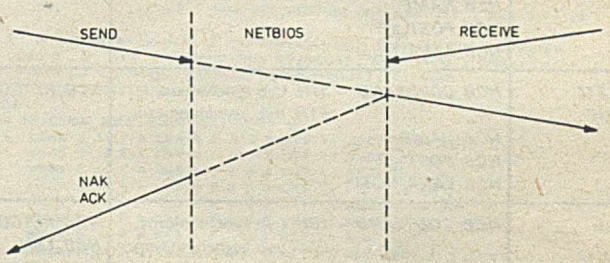
Bezpołączeniowy sposób współpracy można ustalić określając maksymalną częstotliwość transmisji. Wtedy odbiornik musi być przygotowany zawsze do odbioru po upływie pewnego kwantu czasowego. Przesyłanie bezpołączeniowe ma praktyczne zastosowanie zarówno dla łączy wielopunktowych, jak i dwupunktowych.

W NETBIOS-ie są dwa polecenia, *CALL* i *LISTEN*, umożliwiające nawiązanie połączenia. Polecenie *LISTEN* ma zasięg lokalny, a *CALL* sięga przez oprogramowanie komunikacyjne do stanu polecenia *LISTEN* i na tej podstawie podejmuje transmisję lub nie (rys. 1). Przez *LISTEN* można nawiązać połączenie z wybranym bądź dowolnym procesem.



Rys. 1. Ilustracja nawiązania połączenia

Przesyłanie połączeniowe w sieci odbywa się przy wykorzystaniu poleceń *SEND* i *RECEIVE*, przy czym polecenia *RECEIVE* mają zasięg lokalny (rys. 2). Przesyłanie połączeniowe ma zastosowanie jedynie dla łączy dwupunktowych.



Rys. 2. Ilustracja przesyłania połączeniowego w trybie potwierdzeń

Polecenia nadawania i odbioru wiadomości mogą być użyte w trybie synchronicznym i asynchronicznym. Tryb synchroniczny zapewnia pełną synchronizację, jednak jego wadą jest długi czas oczekiwania oraz możliwość zaginięcia informacji dla poleceń *RECEIVE*. Praca synchroniczna jest typowa dla systemów wielozada-

Tabela 1. Funkcje przerwania 2A

Znaczenie	Parametry wejściowe	Parametry wyjściowe
Sprawdzenie instalacji sprzęgu	*AH=00H*	*AH=00H* – nie zainstalowany *AH≠00H* – zainstalowany
Sprawdzenie bezpośredniego dostępu do zasobów dyskowych	*AX=0300H*	Znacznik przeniesienia: *C=1* – brak dostępu *C=0* – potwierdzenie dostępu
Wykonanie polecenia NETBIOS-a z obsługą pewnych błędów (*09H, 12H, 21H*); w wypadku błędu wykonanie polecenia jest ponawiane wiele razy	*AX=0400H* *ES:BX* – adres pola *NCB*	*AH=00H* i *AL=00H* – polecenie wykonane poprawnie *AH=01H* i *AL=* (kod błędu) – wystąpienie błędu podczas wykonywania polecenia
Wykonanie polecenia NETBIOS-a bez obsługi błędów	*AX=0401H* *ES:BX* – adres pola NCB	*AH=00H* i *AL=00H* – polecenie wykonane poprawnie *AH=01H* i *AL=*(kod błędu) – wystąpienie błędu podczas wykonywania polecenia
Odczytanie parametrów sieci po zainstalowaniu programu IBM PC LAN	*AX=0500H*	*AX* – zarezerwowany *BX* – liczba dostępnych nazw do zadeklarowania *CX* – liczba dostępnych poleceń NETBIOS-a *DX* – liczba sesji możliwych do zestawienia

Tabela 2. Zestawienie parametrów dla poleceń NETBIOS-a ([1] – pole określające adapter sieci realizujący polecenia, [2] – pole używane przy pracy asynchronicznej, [3] – ustawienie zera powoduje wyłączenie działania przeterminowania, odpowiednio dla nadawania lub odbioru)

Polecenie	Parametry wejściowe	Parametry wyjściowe
RESET	*NCB_COMMAND* – *32H*, tryb synchroniczny *NCB_NUM* – liczby obsługiwanych bloków NCB *NCB_LSN* – liczba obsługiwanych sesji *NCB_LANA_NUM*[1]	*NCB_RETCODE*
CANCEL	*NCB_COMMAND* – *35H*, tryb synchroniczny *NCB_LANA_NUM*[1] *NCB_BUFFER* ⓐ – adres odwoływanego bloku NCB	*NCB_RETCODE*
ADAPTER STATUS	*NCB_COMMAND* – *33H*, tryb synchroniczny, *B3H*, tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_CALLNAME* (p. opis polecenia *ADAPTER STATUS*) *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH*
ADD NAME	*NCB_COMMAND* – *30H*, tryb synchroniczny, *B0H*, tryb asynchroniczny *NCB_NAME* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_NUM*
ADD GROUP NAME	*NCB_COMMAND* – *36 H*, tryb synchroniczny, *B6H*, tryb asynchroniczny *NCB_NAME* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_NUM*
DELETE NAME	*NCB_COMMAND* – *31H*, tryb synchroniczny, *B1H*, tryb asynchroniczny *NCB_NAME* *NCB_POST*ⓐ[2] NCB_LANA_NUM[1]	*NCB_RETCODE*
CALL	*NCB_COMMAND* – *10H*, tryb synchroniczny, *90H*, tryb asynchroniczny *NCB_CALLNAME* *NCB_NAME* *NCB_RTO*[3] *NCB_STO*[3] *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LSN*
LISTEN	*NCB_COMMAND* – *11H*, tryb synchroniczny, *91H*, tryb asynchroniczny *NCB_CALLNAME* *NCB_NAME* *NCB_RTO*[3] *NCB_STO*[3] *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LSN* *NCB_CALLNAME* (p. opis polecenia *LISTEN*)
HANG UP	*NCB_COMMAND* – *12H*, tryb synchroniczny, *92H*, tryb asynchroniczny *NCB_LSN* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE*
SEND	*NCB_COMMAND* – *14H*, tryb synchroniczny, *94H*, tryb asynchroniczny *NCB_LSN* *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE*
CHAIN SEND	*NCB_COMMAND* – *17H*, tryb synchroniczny, *97H*,tryb asynchroniczny *NCB_LSN* *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1] *NCB_CALLNAME* (p. opis polecenia *CHAIN SEND*) *1) NCB_LENGTH2 DW 000H* *2) NCB_BUFFER2*ⓐ*DD 00000000H*	*NCB_RETCODE*
RECEIVE	*NCB_COMMAND* – *15H*, tryb synchroniczny, *95H*, tryb asynchroniczny *NCB_LSN* *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH* (liczba bajtów w buforze odbiornika)
RECEIVE ANY	*NCB_COMMAND* – *16H*, tryb synchroniczny, *96H*, tryb asynchroniczny *NCB_NUM* *NCB_BUFFER*ⓐ *NCB_LENGTH* NCB_POSTⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH* *NCB_LSN* *NCB_NUM* (p. opis polecenia *RECEIVE ANY*)
SESSION STATUS	*NCB_COMMAND* – *34H*, tryb synchroniczny, *B4H*, tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_NAME* (p. opis polecenia *SESSION STATUS*) *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH* (liczba bajtów w buforze stanu)
SEND DATAGRAM	*NCB_COMMAND* – *20H*, tryb synchroniczny, *A0H*, tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_NUM* *NCB_CALLNAME* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*	*NCB_RETCODE*
SEND BROADCAST DATAGRAM	*NCB_COMMAND* – *22H*, tryb synchroniczny, *A2H*, tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_NUM* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE*
RECEIVE DATAGRAM	*NCB_COMMAND* – *21H*, tryb synchroniczny, *A1H*,tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH* *NCB_CALLNAME* (p. opis obsługi datagramów) *NCB_NUM*
RECEIVE BROADCAST DATAGRAM	*NCB_COMMAND* – *23H*, tryb synchroniczny, *A3H*, tryb asynchroniczny *NCB_BUFFER*ⓐ *NCB_LENGTH* *NCB_NUM* *NCB_POST*ⓐ[2] *NCB_LANA_NUM*[1]	*NCB_RETCODE* *NCB_LENGTH* *NCB_CALLNAME* (p. opis obsługi datagramów)

niowych. Zawiadomienie o zakończeniu procesu jest uzyskiwane na zasadzie odpytywania (ang. *polling*), a nie przez przerwania. Tryb asynchroniczny nie ma wad pracy synchronicznej, jednak stwarza kłopoty przy sterowaniu. Praca asynchroniczna jest typowa dla systemów jednozadaniowych, wykorzystujących przerwania.

Po zainstalowaniu oprogramowania sieciowego IBM PC LAN, programy użytkowe należy sprzęgać z NETBIOS-em korzystając z przerwania *2AH* (tab. 1). Przerwanie *5CH* jest ściśle związane z adapterem sieci i powoduje uzależnienie się od sprzętu, natomiast korzystając z przerwania *2AH* uniezależniamy się od rozwiązania sprzętowego w przyszłych zastosowaniach. Dla przerwania *2AH* nie ma możliwości korzystania z poleceń: *RESET, ADD GROUP NAME, SEND BROADCAST DATAGRAM* i *RECEIVE BROADCAST DATAGRAM*.

Sposób wykorzystania pól w bloku *NCB* dla poszczególnych poleceń NETBIOS-a, przedstawiono w tabeli 2.

HENRYK GIZDOŃ
ADAM PAWLAK
WŁODZIMIERZ WRONA
Instytut Elektroniki
Politechnika Śląska
Gliwice

Język opisu VHDL – podstawowe mechanizmy (2)

W dalszej części opisu języka VHDL przedstawiono następujące mechanizmy i pojęcia języka: parametry formalne, podprogramy, pakiety, atrybuty i dyrektywy. Pojęcie pakietu, ze względu na interesujące możliwości jego wykorzystania omówiono szerzej, z podaniem przykładowego rozwiązania zawierającego deklaracje związane z logiką czterowartościową. Dodatkowo przedstawiono mechanizm upływu czasu wraz z jego deklaracją.

PARAMETRY FORMALNE

Jednostki projektowe mogą mieć różne parametry o wartościach zależnych od środowiska, w którym następuje konkretyzacja nowego składnika. W języku VHDL parametry formalne można przypisywać jednostce projektowej przez użycie słowa zastrzeżonego *generic* z wyszczególnieniem wszystkich jej parametrów. Przykładem może być deklaracja i konkretyzacja jednostki projektowej bramki sumy logicznej, gdzie *X_COORD* i *Y_COORD* są współrzędnymi bramki w projekcie rozmieszczenia układu (ang. *placement*). Typy *X_INTEGER* i *Y_INTEGER* deklarują liczby całkowite określające zakresy współrzędnych:

```
-- deklaracja sprzęgu bramki OR
entity OR_GATE
-- deklaracja portów
generic
  (X_COORD : X_INTEGER, Y_COORD : Y_INTEGER)
end OR_GATE;
-- instrukcja konkretyzacji
OR1: OR_GATE port (S2, S3, COUT);
             generic (20, 35);
```

PODPROGRAMY

W języku VHDL podprogramy znajdują zastosowanie do realizacji opisów algorytmicznych. Są one realizowane w zerowym czasie symulacji, tzn. ich wykonanie nie wnosi dodatkowych opóźnień do czasu symulacji. Istnieją dwa rodzaje podprogramów: funkcje i procedury. Funkcje służą przede wszystkim do obliczeń. Zadaniem funkcji jest m. in. wyznaczanie wartości sygnałów wspólnej magistrali. Należy zaznaczyć, że funkcja po zakończeniu obliczeń przekazuje pojedynczą wartość, lecz wartość ta może być typu złożonego. Przykładem może być deklaracja funkcji, zamieniającej daną typu *BIT_VECTOR* na liczbę naturalną, oraz jej wywołanie (wydruk 1).

```
-------------------------------------------------------------
function WART_NAT(DANE : BIT_VECTOR) return NATURAL is
    variable NATURAL := 0;
begin
    for I::DANE'LEFT to DANE'RIGHT loop
        A := A*2 + BIT'POS(DANE(I));
    end loop;
    return A;
end WART_NAT;
-------------------------------------------------------------
```

Wydruk 1

Procedura jest podprogramem, który może być traktowany jako krótki zapis sekwencji instrukcji. Procedura specyfikuje określony algorytm. Może być używana do dekomponowania złożonych opisów zachowania na większą liczbę mniejszych modułów. Przykładem może być deklaracja procedury obliczającej wynik operacji i wartości wskaźników jednostki arytmetyczno-logicznej *JAL* (wydruk 2). Instrukcji **case** przyporządkowano zmienną *ZMIENNA_3BIT* z atrybutem będącym liczbą naturalną. Liczba ta jest rezultatem przekształcenia wektora 3-bitowego na liczbę naturalną.

```
-------------------------------------------------------------
procedure WYNIK_JAL (R,S,OP : in BIT_VECTOR; Cn : in BIT;
                     F : out BIT_VECTOR; Cn4,P,G,OVF : out BIT) is

-- R, S - dane wejsciowe, F - rezultat, Cn - przeniesienie wejsciowe
-- Cn4, P, G, OVF - przeniesienie wyjsciowe, propagacja i generacja
-- (wskaźniki dla układu szybkich przeniesiczeń), nadmiar
-- OP - kod realizowanej operacji
begin
case ZMIENNA_3BIT'(WART_NAT(OP)) is
      when 0 => F & Cn4 & P & G & OVF <= ADDF(R,S,Cn);
      when 1 => F & Cn4 & P & G & OVF <= SUBF(S,R,Cn);
      when 2 => F & Cn4 & P & G & OVF <= SUBF(R,S,Cn);
      when 3 => F & Cn4 & P & G & OVF <= ORF(R,S,Cn);
      when 4 => F & Cn4 & P & G & OVF <= ANDF(R,S,Cn);
      when 5 => F & Cn4 & P & G & OVF <= ANDF(not R,S,Cn);
      when 6 => F & Cn4 & P & G & OVF <= XORF(R,S,Cn);
      when 7 => F & Cn4 & P & G & OVF <= XORF(not R,S,Cn);
end case;
end WYNIK_JAL;
-------------------------------------------------------------
```

Wydruk 2

PAKIETY

Mechanizmem grupującym i przechowującym deklaracje typów, podtypów, atrybutów, stałych, podprogramów i składników jest pakiet. Z deklaracji zawartych w pakietach można wielokrotnie korzystać przy realizacji nowych projektów. Przykładem może być pakiet zawierający deklaracje związane z logiką czterowartościową (wydruk 3).

```
-------------------------------------------------------------
package LOG_4WART is
-- definicja zbioru elementów logiki czterowartościowej
    type LOG_4 is
          ('0'    -- stan niski
           '1'    -- stan wysoki
           'Z'    -- stan dużej impedancji
           'E'    -- więcej niż jedno źródło sygnałów dołączone do
          );      -- wspólnego węzła przyjmuje stan różny od 'Z'
-- definicja wektora elementów logiki czterowartościowej
    type LOG_4VECTOR is array (NATURAL range <>) of LOG_4;
-- funkcja realizująca konwersję między wektorami typu LOG_4 i BIT
    function CZYTAJ_WE (BUS : LOG_4VECTOR) return BIT_VECTOR is
-- funkcja realizująca konwersję między wektorami typu BIT i LOG_4
    function ZAPISZ_WY (BUS : BIT_VECTOR) return LOG_4VECTOR is
-- funkcja arbitra magistrali generująca warunek błędu, jeżeli więcej niż
-- jedno źródło sygnału sterującego magistralą przyjmie stan różny od 'Z'
    function KONFLIKT_WY (SOURCES : LOG_4VECTOR) return LOG_4 is
-- deklaracje typów trójstanowych związanych z funkcją arbitra magistrali
    type BIT_3STAN is atomic KONFLIKT_WY LOG_4;
    type VECTOR_3STAN is array (NATURAL range <>)
                              of atomic KONFLIKT_WY LOG_4;
-- deklaracja zakresu obciążalności portów i sygnałów
    subtype ZAKRES is INTEGER range 1 to 30;
-- deklaracja atrybutu obciążalności portów i sygnałów
    attribute FANOUT of port, signal is ZAKRES;
end LOG_4WART;
-------------------------------------------------------------
```

Wydruk 3

W pakiecie tym funkcja *CZYTAJ_WE* realizuje konwersję między wektorami bitów zadeklarowanych w logice czterowartościowej

i dwuwartościowej. Założono, że stany *Z* i *E* są przez tę funkcję odwzorowane na 0. Odwzorowanie stanu *E* na 0 jest uzasadnione, ponieważ stan *E* będzie na ogół wykrywany w założeniach i pojawienie się takiego stanu spowoduje wygenerowanie komunikatu o błędzie. Odwzorowanie stanu *Z* na stan 0 wiąże się z przyjęciem założenia, że układ fizyczny w ten właśnie sposób interpretuje stan dużej impedancji. W ogólnym wypadku można by zdefiniować funkcję, która oprócz wartości wyjściowego wektora bitów przekaże wskaźnik błędu, jeśli wystąpią stany *Z* i *E* w wektorze wejściowym.

Funkcja arbitra magistrali *KONFLIKT_WY* wykrywa konflikt na magistrali trójstanowej, tzn. sytuację, w której na magistralę zostaną jednocześnie podane sygnały różne od *Z* (co najmniej z dwóch źródeł). Funkcja ta została związana z deklaracjami typów *VECTOR_3STAN* i *BIT_3STAN* przez użycie słowa zastrzeżonego **atomic** (deklarującego typ sygnału jednolitego). Dodatkowo w pakiecie zamieszczono definicję atrybutu obciążalności wyjść *FAN-OUT*.

Zawartość raz zdefiniowanego pakietu jest dostępna w innych opisach przez klauzule **with** i **use** umieszczone na początku opisów. Przykładem użycia klauzul **with** i **use** może być deklaracja sprzęgu multipleksera z wyjściem trójstanowym korzystająca z pakietu *LOG_4WART:*

```
with package LOG_4WART; use LOG_4WART;
 entity MULTIPLEK_TS
   -- deklaracja sprzęgu
   is
 end MULTIPLEK_TS;
```

Klauzula **with** umożliwia dostęp do zadeklarowanych pozycji pakietu bibliotecznego pośrednio przez nazwę pakietu, kropkę i nazwę pozycji (np. *LOG_4WART.CZYTAJ_WE(DANE)*). Klauzula **use** umożliwia dostęp do wybranej pozycji pakietu bezpośrednio tylko przez podanie jej nazwy. Definiując nowy pakiet za pomocą klauzuli **with** i **use** można odwoływać się i używać deklaracji z innych pakietów.

ATRYBUTY

VHDL zawiera mechanizm umożliwiający wyrażenie dodatkowych cech i właściwości wybranych jednostek języka oraz nadawanie tym jednostkom atrybutów. Atrybut jest wartością, która towarzysząc nazwie jednostki opisuje jedną z charakterystycznych cech tej jednostki.

Atrybut ma nazwę i typ. Może on występować razem ze sprzęgiem, ciałem, pakietem, podprogramem, stałą, zmienną, sygnałem, etykietą lub typem. Ten sam atrybut może być związany z różnymi jednostkami lub nawet z różnymi rodzajami jednostek. Na przykład atrybut *LAST_VALUE* zdefiniowany pierwotnie w języku towarzyszy portom jednostek projektowych oraz wszystkim sygnałom. Sam atrybut nie ma wartości, lecz przyjmuje wartość tylko wtedy, kiedy jest związany z wybraną jednostką. Wartość atrybutu jest wówczas wyrażana przez nazwę jednostki zapisaną przed nazwą atrybutu. Na przykład, wartość atrybutu *LAST_VALUE* dla sygnału *S* można zapisać:

S'LAST_VALUE

Podobnie można zapisać atrybut ciała określający, czy dane ciało zawiera instrukcje konkretyzacji składnika:

B'Childless

VHDL zawiera bogaty repertuar atrybutów zdefiniowanych pierwotnie wyrażonych przez wartości, funkcje, typy, zakresy i sygnały. Przykładem atrybutów zdefiniowanych pierwotnie są:
- wartości *LEFT, RIGHT, HIGH, LOW* dla typów skalarnych, oznaczające odpowiednio lewą, prawą, górną i dolną granicę typu,
- funkcje *LEFT, RIGHT, HIGH, LOW* dla tablic obiektów, oznaczające granice indeksów typów tablicowych,
- funkcje *RANGE* i *LENGTH*, oznaczające zakres i długość tablicy,
- wartość *LAST_VALUE*, oznaczająca ostatnią wartość związaną z portem lub sygnałem,
- sygnał *STABLE [T]*, umożliwiający określenie stałości wybranego sygnału w okresie *T* jednostek czasu.

Atrybuty mogą być również definiowane przez użytkownika, są one wówczas ograniczone do zmiennych. Przykładem zadeklarowanego atrybutu może być atrybut o nazwie *POZIOM* (realizacja projektu):

```
type POZIOMY is (UKŁADOWY, LOGICZNY, FUNKCJONALNY,
SYSTEMOWY);
attribute POZIOM of entity, body is POZIOMY;
```

Wykorzystanie zdefiniowanego atrybutu polega na podaniu specyfikacji wartości atrybutu dla wybranej jednostki programowej, np.:

```
for POZIOM of entity MULTIPLEK_TS use LOGICZNY;
```

DYREKTYWY

Dyrektywy są mechanizmem umożliwiającym wprowadzenie do programu pewnej dodatkowej informacji, potrzebnej w trakcie jego realizacji. Rozróżnia się następujące dyrektywy: *assert, import, initialize* oraz *select*.

Dyrektywa *assert* jest równoważna współbieżnej instrukcji komunikatu. Składa się – podobnie jak ta instrukcja – z warunku logicznego, raportu oraz klauzuli ważności (ang. *severity*). Dyrektywa *assert* może występować w sprzęgu jednostki projektowej, towarzysząc w ten sposób wszystkim instrukcjom bloku ciał związanych z danym sprzęgiem.

Dyrektywa *import* zwiększa możliwości wykorzystania deklaracji występujących w ciele jednostki projektowej na konfigurację danego ciała. W dyrektywie *import* mogą występować nazwy portów, parametrów formalnych (rodzajowych), stałych, typów, podtypów i atrybutów.

Dyrektywa *initialize* umożliwia ustalenie wartości początkowych obiektów określonego typu. Wszystkie wyrażenia tej dyrektywy muszą mieć ustaloną wartość.

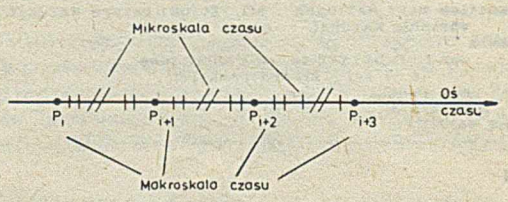
Dyrektywa *select* przyporządkowuje wybraną konfigurację do instrukcji bloku lub konkretyzowanego składnika, dla których specyfikacja konfiguracji nie istnieje. Dyrektywa ta ogranicza możliwość wyboru konfiguracji.

MODELOWANIE UPŁYWU CZASU – DEKLARACJA CZASU

W opisie zachowania projektu układu cyfrowego ważnym zagadnieniem jest modelowanie czasu. Zależności czasowe w opisie projektu układu są reprezentowane jako opóźnienia między zmianą wartości sygnałów wejściowych, a zmianą sygnałów wyjściowych. Opóźnienie jest związane z bieżącym czasem symulacji. Język musi również reprezentować współbieżność operacji w sprzęcie.

Model czasowy w języku VHDL (podobnie jak w języku opisu sprzętu Conlan) składa się z makroskali i mikroskali (rys.). Makroskala reprezentuje czas rzeczywisty i jest mierzona w dyskretnych jednostkach czasu. W mikroskali jednostkowe opóźnienie jest wielkością nieskończenie małą. Między dwiema makrojednostkami czasu może istnieć dowolna liczba mikrojednostek czasu. Zadeklarowany pierwotnie typ fizykalny *TIME* ułatwia posługiwanie się jednostkami czasu:

```
type TIME is range 0 to 1E20;
 units
```



Model czasu w VHDL

```
    fs;                    -- femtosekunda
    ps = 1000 fs;          -- pikosekunda
    ns = 1000 ps;          -- nanosekunda
    μs = 1000 ns;          -- mikrosekunda
    ms = 1000 μs;          -- milisekunda
    s = 1000 ms;           -- sekunda
    min = 60 s;            -- minuta
    hr = 60 min;           -- godzina
end units;
```

Typ fizykalny *TIME* jest traktowany jako typ *INTEGER*, z towarzyszącym zbiorem jednostek czasu. Pierwsza jednostka jest wyróżniona jako podstawowa. Pozostałe jednostki w zbiorze są reprezentowane jako wielokrotność jednostki podstawowej. W typie *TIME* jednostką podstawową jest femtosekunda (fs). Każda z pozostałych jednostek jest wielokrotnością fęmtosekundy. Wprowadzanie zbioru jednostek pozwala zapisywać czas w sposób naturalny, np. 5 ns, 6,5 ms, 81,2 ps. Taka specyfikacja jest automatycznie transponowana na wielokrotność femtosekundy, tj. 5 000 000 fs, 6 500 000 000 000 fs, 81 200 fs.

W opisie składników specyfikuje się ustaloną zależność między przeszłą i obecną wartością wejść składników a przyszłą wartością ich wyjść; oznacza to, że dla zbioru wartości wejść, opis w języku VHDL przewiduje możliwość określenia spodziewanych wartości wyjść w zdefiniowanym momencie w przyszłości. W trakcie upływu czasu projektowane wartości wyjść stają się wartościami aktualnymi i są przesyłane do wejść innych składników przez port i sygnały łączące składniki.

* * *

Język VHDL wykorzystuje dorobek języków programowania, a w szczególności języka Ada. Pewne mechanizmy tj. typy, pakiety, atrybuty, podprogramy zdefiniowane w Adzie są również bezpośrednio realizowane w języku VHDL. Klasa dostępnych obiektów została rozbudowana o nowe pojęcie sygnału i jego nośnika, co umożliwiło reprezentację połączeń jednostek projektowych i ich składników. Porównując instrukcje języka VHDL z instrukcjami innych języków programowania można stwierdzić, że instrukcje sekwencyjne VHDL a w tym wszystkie instrukcje sterujące, są podstawowymi instrukcjami innych języków programowania (np. Ady, C, Moduli-2). Instrukcje języka VHDL, opisujące współbieżne operacje w sprzęcie, zasadniczo różnią się sposobem zapisu, w odniesieniu do innych języków. W języku VHDL stosuje się instrukcje bloku i procesu, a np. w Adzie do tego samego celu służą zadania (ang. *tasks*). Deklaracja komunikatów języka VHDL różni się od deklaracji wyjątków Ady sposobem zapisu, spełnia jednak podobną rolę.

Wersja 7.2 języka VHDL nie zawiera wszystkich mechanizmów Ady, umożliwia jednak definiowanie abstrakcyjnych opisów zachowania złożonych układów VLSL, w początkowym etapie ich projektowania. Na poziomach przesłań międzyrejestrowych oraz logicznym umożliwia efektywne wyrażanie wszystkich niezbędnych aspektów opisu sprzętu.

Uproszczone specyfikowanie...

dokończenie ze s. 4

Jak łączy się specyfikacje indywidualnych składowych w specyfikację całego systemu?

Formalnie, stwierdzenie że system M jest złożeniem dwóch systemów M_1 i M_2 znaczy, iż specyfikacja M, która jest formułą logiki temporalnej, jest koniunkcją specyfikacji M_1 i M_2. W metodzie aksjomatu przejść, specyfikację M otrzymuje się przez zwykłe połączenie specyfikacji M_1 i M_2. Funkcje stanu w specyfikacji M składają się z funkcji stanu ze specyfikacji M_1 i M_2, a zbiór akcji specyfikacji M jest sumą zbiorów akcji specyfikacji M_1 i M_2.

Połączenie specyfikacji może wymagać pewnego przemianowania. Być może zajdzie potrzeba przemianowania wewnętrznych funkcji stanu i akcji, aby uniknąć konfliktów, ponieważ wewnętrzne funkcje stanu specyfikacji M_1 reprezentują funkcje stanu różne od wewnętrznych funkcji stanu specyfikacji M_2. Ponadto, czynność połączenia specyfikacji M_1 i M_2 może wiązać się z przemianowaniem lub identyfikacją funkcji stanu sprzężenia. Przykładowo, M_1 może być specyfikacją obwodu z funkcją stanu sprzężenia nazwaną **output**, która reprezentuje napięcie na jego wyjściu, a M_2 – specyfikacją obwodu z funkcją stanu sprzężenia nazwaną **input**, która reprezentuje napięcie na jego wejściu. Połączenie wyjścia systemu M_1 z wejściem systemu M_2 oznacza, że **input** i **output** stają się dwiema nazwami tej samej funkcji stanu.

☆

Chciałbym podziękować Amirowi Pnueli za nauczenie mnie, co oznacza kwantyfikacja egzystencjalna nad funkcjami stanu, a Jimowi Hormingowi i Johnowi Guttagowi – za liczne i owocne dyskusje. Następujące osoby niech raczą przyjąć podziękowania za komentarze dotyczące wcześniejszych wersji tego artykułu: Jim Gray, Brant Hailpern, Luigi Logrippo, Jay Misra, Susan Owicki, Willem-Paul de Roever, Fred Schneider, Fritz Vogt, Jeanette Wing i Pamela Zave.

Tłumaczył i opracował
JANUSZ ZALEWSKI

LITERATURA

[1] Alpern B., Schneider B. F.: Verifying Temporal Properties without Using Temporal Logic. Technical Report TR85-723, Department of Computer Science, Cornell University, December 1985

[2] Lamport L.: Reasoning about nonatomic operations. Proc. 10 th Ann. Symp. on Principles of Programming Languages, ACM SIGACT-SIGPLAN, pp. 28-37, January 1983

[3] Lamport L.: „Sometime" is sometimes „not never" – a tutorial on the temporal logic of programs. 7 th Ann. Symp. on Principles of Programming Languages, ACM SIGACT-SIGPLAN, pp. 174-185, January 1980

[4] Milner R.: A Calculus of Communicating Systems. Springer-Verlag, Berlin 1980

[5] Owicki S., Gries D.: An axiomatic proof technique for parallel programs. Acta Informatica, Vol. 6, No. 4, pp. 319-340, 1976

[6] Owicki S., Lamport L.: Proving liveness properties of concurrent programs. ACM Trans. on Programming Languages and Systems. Vol. 4, No. 3, pp. 455-495, July 1982.

Perspektywy rozwoju systemu Unix

Według badań rynkowych przeprowadzonych przez Diebolta, w roku 1990 co czwarty system dla wielu użytkowników będzie oparty na systemie Unix. Prognozy IDC mówią o ponad 700 tysiącach systemów w tym samym roku. Detaquest natomiast podaje wartość około 7 mld dolarów w 1990 r. za oprogramowanie pochodzące od tego systemu.

W 1984 r. było około 120 tys. instalacji Unixa, średnio z sześcioma stanowiskami w systemie. W tym samym czasie było dziesięciokrotnie więcej instalacji pracujących pod kontrolą systemu MS DOS. W następnych latach udział instalacji z systemem Unix zwiększał się, osiągając 1/6 w 1986 r. i prawdopodobnie 1/5 (odpowiednio 2 mln Unix i 10 mln MS DOS) w 1990 r. Jednocześnie średnia liczba stanowisk w systemie maleje do 5.

W ramach systemu Xenix opracowano trzy niezależnie sprzedawane moduły: system wykonawczy (Run Time), system wspomagania (Development System) oraz przetwarzania tekstów (Text Processing System). Pełny system zajmuje 9,2 MB, a do właściwego działania potrzebuje 12 MB na dysku. System wykonawczy obejmuje wszystkie polecenia bieżące, a także umożliwia dołączanie nowych użytkowników, urządzeń zewnętrznych i ochronę pamięci; zajmuje 4,1 MB. Wersja Xenix dla komputerów osobistych IBM jest we Francji sprzedawana przez firmę Axis Digital w cenie 7 tysięcy franków. Tyle samo kosztuje system wspomagania (3,8 MB), a przetwarzanie tekstów (1,3 MB) – 2500 franków. Niezależnie opracowano system Xenix 286 przeznaczony dla rodziny mikroprocesorów iAPX 286 i pakietów równoważnych. Jest to wielozadaniowy system dla wielu użytkowników, który przy stosowaniu kostki 80286 pozwala osiągnąć szybkość 3-5 razy większą niż przy poprzednich systemach Xenix dla mikroprocesora 8086. Został opracowany przez Microsoft i Intel na podstawie wersji 7 Unixa firmy Western Electric; zrealizowano w nim takie usprawnienia, jak blokowanie rekordów i plików oraz zabezpieczenie informacji na dysku przy wyłączeniu zasilania. **J.R**

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Turbo C

Wywoływanie funkcji, operacje na rejestrach, wstawki asemblerowe

Każde opracowanie wywołania funkcji składa się z trzech czynności:
- wykonania prologu wywołania funkcji,
- wykonania funkcji,
- wykonania epilogu wywołania funkcji.

W prologu wywołania funkcji są opracowywane jej argumenty i umieszczane na stosie, w kolejności od ostatniego do pierwszego. Bezpośrednio po tym, na stosie jest umieszczany ślad wywołania funkcji. W wypadku funkcji bliskich ślad ten jest 16-bitowy, a w wypadku funkcji dalekich i odległych 32-bitowy.

W epilogu wywołania funkcji usuwa się ze stosu umieszczone tam argumenty (usunięcie śladu następuje w epilogu funkcji). Tym samym, w odróżnieniu od implementacji innych języków programowania, usunięciem argumentów ze stosu nie zajmuje się funkcja wywoływana, lecz funkcja wywołująca. Powoduje to pewne wydłużenie kodu programu wynikowego, ale umożliwia posługiwanie się funkcjami ze zmienną liczbą argumentów.

W prologu wykonania funkcji zapamiętuje się na stosie zawartość niektórych rejestrów oraz rezerwuje miejsce dla zmiennych automatycznych. W epilogu wykonania funkcji odtwarza się zapamiętane zawartości rejestrów i zdejmuje ze stosu ślad wywołania funkcji. Jeśli funkcja jest zadeklarowana z modyfikatorem *interrupt*, to zapamiętanie i odtworzenie dotyczy dodatkowo rejestrów *AX, BX, CX, DX, DS* i *ES*. W opisach implementacji nie ujawniono zasad zapamiętywania i odtwarzania zawartości rejestrów. Zagwarantowano jedynie, że wywołanie funkcji nie spowoduje zmiany zawartości rejestrów *BP, CS, SI* i *DI*.

Przykład

```
#include <stdio.h>
main()
{
  unsigned Num;
  long unsigned Fact(unsigned);

  printf("Argument = ");
  scanf("%u", &Num);
  printf("\n%u! = %lu", Num,Fact(Num));
}

long unsigned
Fact(unsigned Par)
{
  if(Par > 1)
    return Par * Fact(Par - 1);
  return 1;
}
```

Wykonanie programu powoduje wyznaczenie silni zadanej liczby. W modelu *Small* każde wywołanie funkcji *Fact* powoduje zarezerwowanie na stosie sześciu bajtów. W liczbie tej, rejestr *BP*, ślad i argument wywołania zajmują po 2 bajty. W modelu *Large* każde wywołanie zajmuje 8 bajtów.

Rezultaty funkcji są zazwyczaj pozostawiane w rejestrach procesora. Dane 8- i 16-bitowe (*char, short, int, enum* i wskazania kategorii *near*) są pozostawiane w rejestrze *AX*, natomiast dane 32-bitowe (*long* oraz wskazania kategorii *far* i *huge*) są pozostawiane w rejestrach *AX* (część mniej znacząca) i *DX* (część bardziej znacząca). Wyjątek od tej zasady dotyczy danych zmiennopozycyjnych *float* i *double*. Są one pozostawiane na szczycie stosu koprocesora albo emulatora.

Przykład

```
#include <stdio.h>

long unsigned Fix = 65536 + 13;

main()
{
  printf("%u == %d", (unsigned)Fix,CutOff(Fix));
}

CutOff(long Par)
{
  return Par;
}
```

Wykonanie programu powoduje wyprowadzenie napisu 13==13. Mimo iż program jest błędny, celowe jest przemyślenie przebiegu jego wykonania.

Wykaz nazw i typów pseudozmiennych

Nazwa	Typ	Przeznaczenie
_AX	unsigned int	Akumulator
_AL	unsigned char	
_AH	unsigned char	
_BX	unsigned int	Rejestr indeksowy
_BL	unsigned char	
_BH	unsigned char	
_CX	unsigned int	Licznik cykli
_CL	unsigned char	
_CH	unsigned char	
_DX	unsigned int	Rejestr danych
_DL	unsigned char	
_DH	unsigned char	
_CS	unsigned int	Adres segmentu kodu
_DS	unsigned int	Adres segmentu danych
_SS	unsigned int	Adres segmentu stosu
_ES	unsigned int	Adres segmentu pomocniczego
_SP	unsigned int	Odstęp względem SS
_BP	unsigned int	Odstęp względem SS
_DI	unsigned int	Zmienna rejestrowa
_SI	unsigned int	Zmienna rejestrowa

Struktury zajmujące 16 bitów są pozostawiane w rejestrze *AX*, a struktury zajmujące 32 bity są pozostawiane w rejestrach *AX* (część mniej znacząca) i *DX* (część bardziej znacząca). Struktury wymagające więcej miejsca są pozostawiane w statycznym obszarze pamięci. W rejestrze *AX* (modele *Tiny, Small, Medium*) albo w rejestrach *AX* i *DX* (modele *Compact, Large* i *Huge*) jest pozostawiane wskazanie adresowe tego obszaru.

W tych rzadkich wypadkach, gdy jest pożądane bezpośrednie odwołanie się do rejestrów procesora, można posłużyć się nazwami pseudozmiennych. Mnemoniki tych nazw ściśle odpowiadają mnemonikom nazw rejestrów i różnią się od nich tylko tym, iż rozpoczynają się od znaku podkreślenia. W tabeli podano pełen wykaz nazw i typów pseudozmiennych.

Wymienione pseudozmienne mogą być traktowane dokładnie tak, jak pozostałe zmienne programu. Należy jednak mieć na uwadze, że ponieważ program wynikowy intensywnie korzysta z rejestrów procesora, nieprzemyślane zmiany zawartości tych rejestrów mogą spowodować załamanie się wykonywania programu. Uwaga ta dotyczy w szczególności takich rejestrów jak *CS, SS, SP* i *BP*. Nie dotyczy ona oczywiście rejestrów nie używanych przez program wynikowy, a także niektórych rejestrów o zawartościach zapamiętywanych i odtwarzanych podczas wywoływania funkcji.

Przykład

```
# include <stdio.h>
# include <dos.h>
main ()
{
 char Chr;
 printf ("Press any combination of keys");
 _AH = 7;
 geninterrupt (0x21);
 Chr = _AL;
 printf ("\n\nYou entered");
 if (Chr) {
    printf ("ASCII character");
    if (Chr < 32)
       printf ("Ctrl-%c", (chr & 31) + '@');
       else
       putchar (Chr);
  } else {
    _AH = 7;
    geninterrupt (0x21);
    Chr = _AL;
    printf ("non ASCII character, Scan code = %d", Chr);
  }
}
```

Wykonanie programu powoduje wyprowadzenie podpowiedzi (ang. *prompt*) do naciśnięcia dowolnego klawisza albo układu klawiszy (np. *Alt-F9, Ctrl-Z*, itp.). W odpowiedzi na tę akcję zostaje wyprowadzona informacja o kodzie wprowadzonego znaku.

W sposób zbliżony do pseudozmiennych mogą być używane modyfikatory adresowania: *_cs, _ds, _ss* i *_es*. Posłużenie się nimi w deklaracji zmiennej wskazującej powoduje przyjęcie domniemania, że w zmiennej jest reprezentowany jedynie odstęp (ang. *offset*) adresu w segmencie, natomiast numer segmentu znajduje się w podanym rejestrze. W szczególności, opracowanie deklaracji

```
long_ds * Ptr;
```

powoduje utworzenie zmiennej, której mogą być przypisywane takie wskazania zmiennych typu (*long int*), których numer segmentu znajduje się w rejestrze *DS*. Wynika stąd oczywiście, że przytoczona deklaracja jest równoważna deklaracji

```
long near *Ptr;
```

Przykład

```
unsigned volatile _es * const Ptr = (unsigned *)0x6C;

main()
{
  _ES = 0x40;
  while(!kbhit())
    printf("%Bu", *Ptr);
  (void)getch();
}
```

Wykonanie programu powoduje sukcesywne wyprowadzanie wartości danych niejawnie przypisywanych zmiennej nietrwałej o adresie *40:6C*, aktualizowanej asynchronicznie i poza programem co około 18,2 milisekundy.

Znacznie większe możliwości niż posługiwanie się pseudozmiennymi stwarza wykorzystanie wstawek asemblerowych. Wstawki takie mogą być jednak wykorzystywane tylko wtedy, gdy program jest tłumaczony poza systemem Turbo C. W takim wypadku należy posłużyć się kompilatorem języka Turbo C, wywoływanym za pomocą dyrektywy

```
tcc opcje zbiory
```

w której *opcje* reprezentują wykaz opcji, a *zbiory* reprezentują wykaz nazw plików. Elementy każdego z wykazów są oddzielone spacjami.

Jeśli tłumaczony program istotnie zawiera wstawki asemblerowe, to kompilator powinien zostać wywołany z opcją *-B*, a w katalogu bieżącym powinien znajdować się makroasembler *MASM* (firmy Microsoft) i konsolidator *TLINK* (firmy Borland).

Niezależnie od tego faktu, nastąpi skompilowanie wszystkich plików wymienionych w dyrektywie *tcc* i skonsolidowanie ich z bibliotekami dobranymi stosownie do modelu pamięci. Przez domniemanie przyjmuje się model Small. Za pomocą opcji *-mx*, w której *x* jest pierwszą literą nazwy modelu, można obrać inny model adresowania pamięci. Z pozostałych opcji na uwagę zasługują:
– *M*, która poleca utworzenie mapy konsolidacji;
– *I katalog*, określająca nazwę katalogu, w którym znajdują się zbiory nagłówkowe;
– *L katalog*, określająca nazwę katalogu, w którym znajdują się biblioteki.
W wypadku nieużycia opcji określającej nazwę katalogu, wymagane pliki będą poszukiwane w katalogu bieżącym.

Przykład

```
tcc -b -Iinclude -Llib source
```

Wykonanie dyrektywy powoduje skompilowanie i skonsolidowanie z bibliotekami pliku *SOURCE.C*, zapewne zawierającego wstawki asemblerowe. Pliki nagłówkowe będą poszukiwane w podkatalogu *INCLUDE*, a biblioteki będą poszukiwane w podkatalogu *LIB*. Program wykonywalny zostanie umieszczony w pliku *SOURCE.EXE*.

Każdą wstawkę asemblerową rozpoczyna słowo kluczowe *asm*, a kończy znak „;" (średnik). Jeśli średnik jest ostatnim znakiem wiersza, to może być pominięty. W danym wierszu można zapisać więcej niż jedną wstawkę asemblerową. Wstawka asemblerowa jest traktowana tak jak instrukcja programu źródłowego.

Przykład

```
if(a > 5){
  asm mov ah,1  ;  asm mov al,2
} else
  asm mov ah,3
  asm mov al,4
```

Opuszczenie nawiasów klamrowych jest niedozwolone. Ostatnia wstawka jest wykonywana bezwarunkowo, gdyż nie należy do instrukcji *if... else*.

Wstawka asemblerowa może mieć postać instrukcji albo deklaracji danych. Jeśli ma postać instrukcji, to po słowie kluczowym *asm* następuje kod operacji i argumenty. Jeśli ma postać deklaracji, to zamiast kodu operacji występuje dyrektywa makroasemblera, taka jak *db, dd, dw* i *extrn*. Wstawki asemblerowe mające postać instrukcji są umieszczone w segmencie *CODE*, a wstawki asemblerowe mające postać deklaracji są umieszczane w segmencie *DATA*.

Wstawki asemblerowe mogą zawierać odwołania do wszystkich zmiennych i etykiet programu, ale nie mogą zawierać własnych etykiet. Jeśli wstawka asemblerowa zawiera odwołanie do nie kwalifikowanego pola struktury, to odwołanie to jest traktowane jak

literał o wartości równej różnicy adresu pola i adresu struktury. W takim wypadku zabrania się, aby dwóm polom o tej samej nazwie odpowiadały różne odstępy.

Przykład

```
#include <stdio.h>

struct{
  unsigned One,Two;
} Str = { 1,2 };

main()
{
  unsigned Sum(void);

  printf("Sum = %u", Sum());
}

unsigned
Sum(void)
{
  asm   lea di,Str
  asm   mov ax,Str.One
  asm   add ax,[di].Two
}
```

Wykonanie programu powoduje wyprowadzenie sumy pól struktury *Str*. Wstawki asemblerowe zostaną przetłumaczone na kod
mov ax, DGROUP:Str.
add ax, [di + 2]
a rezultat funkcji *Sum* zostanie przekazany poprzez rejestr *AX*.

TMK
pakiet syntezy mowy

dokończenie ze s. 12

Prostym przykładem zastosowania pakietu jest bezpośrednie wykorzystanie jego modułów w dyrektywach systemowych. Jeśli plik służący do kompilacji i konsolidacji programów zawiera następujące dyrektywy:

...
LINK plik, plik nul, mojabibl
IF errorlevel 1 gadaj są błędy

to pojawienie się błędu zostanie zasygnalizowane przez wypowiedzenie zdania *są błędy*.

Innym przykładem jest program napisany w języku C (która_godzina·c), który podaje na głos datę i godzinę (p. wydruk).

LITERATURA

[1] Hahn S.: Teoria sygnałów i zakłóceń. Poradnik inżyniera. Radioelektryka. WNT, Warszawa, s. 881–918, 1969
[2] Lee D. L., Lochovsky F. H.: Voice response systems. Computing Surveys, Vol. 15, No. 4, pp. 351–373, 1983
[3] Olaszy G., Podolec G., Fiser J., Poppe A.: Projektowanie systemu syntezy mowy z nieograniczonym słownictwem (w jęz. węgierskim). Inf. Elektron, 21, No. 5, pp. 247–255, 1986
[4] Schroeder M.: Vocoders – Analysis and synthesis of speech. Proc. IEEE, Vol. 54, No. 5, pp. 720–734, 1966
[5] Shichor E., Silberman H. F.: An improved LPC-algorithm for voiced-speech synthesis. IEEE Trans. ASSP, Vol. 32, No. 1, pp. 180–182, 1984
[6] Strube H. W.: Synthesis part of a LOG AREA RATIO Vocesler in analog hardware. IEEE Trans. ASSP, Vol. 25, pp. 381–391, 1977
[7] Tadeuszewicz R.: Głosowe wprowadzanie informacji do komputera. Informatyka, nr 7, s. 5–7, nr 8, s. 12–16, 1987
[8] Yiourgules N., Kokkims A.: Synthesis of Greek vowels using an improved formant synthesizer and reduced memory. pp. 223–228, Digital Techniques in Simulation, Communication and Control. Elsevier Science Publishers, Amsterdam, 1985

Elastyczne środowisko programowania w Adzie

Tworząc środowisko programowe Ady rozszerza się możliwość tego języka. Właściwości takich środowisk wyspecyfikowano w dokumencie o nazwie Stoneman, w szczególności skupiając się na problemach związanych z zarządzaniem i sterowaniem realizacją projektów oprogramowania wielkoskalowego.

Opracowany w firmie Softech Ada Language System (ALS) jest środowiskiem programowym Ady spełniającym wymagania Stonemana, przeznaczonym właśnie do wytwarzania oprogramowania wielkoskalowego. Zestaw narzędzi wspomaga generowanie programów w Adzie, a baza danych przechowuje wszystkie programy, dane i inne informacje generowane podczas użytkowania ALS. Wbudowane programy usługowe pozwalają na wykorzystywanie tego systemu w różnych zestawach sprzętowych.

Omawiane środowisko składa się obecnie z ponad 80 różnych programów narzędziowych, służących programistom w całym okresie istnienia oprogramowania. Narzędzia służące do redagowania tekstów i tworzenia dokumentów pomagają w przygotowaniu specyfikacji i podręczników. Kompilatory, asemblery, konsolidatory, programy uruchomieniowe i inne wspierają fazy kodowania i testowania programów. W fazach projektowania i pielęgnacji korzysta się z obszernego zestawu narzędzi połączonych z bazą danych, służących do sterowania konfiguracją.

PODSTAWOWE WŁAŚCIWOŚCI

Środowisko ALS wykorzystuje jako macierzysty komputer VAX/VMS firmy DEC. Czyni się wysiłki zmierzające do wykorzystania jako komputerów macierzystych maszyn VAX/Unix i DIPS firmy Nippon Telephone and Telegraph. Od pierwszej instalacji, która nastąpiła w styczniu 1985 r., środowisko ALS zainstalowano w ponad 200 miejscach, m. in. w uniwersytetach i laboratoriach rządowych i przemysłowych.

Maszyna docelowa i macierzysta

Programy aplikacyjne tworzone w środowisku ALS, rezydującym na komputerze macierzystym, są wykonywane na odrębnych komputerach docelowych, często wbudowanych w większe środowisko operacyjne, takie jak np. system naprowadzania lotniczego. Pojedynczy system ALS może wspomagać wytwarzanie programów dla dowolnej liczby odmiennych maszyn docelowych. Taka zdolność wytwarzania oprogramowania dla wielu procesorów docelowych w obrębie tego samego środowiska projektowego zwiększa wielokrotnie wielokrotność istniejącego oprogramowania. Powoduje to obniżenie kosztów projektowych i pielęgnacyjnych przez wyeliminowanie powtarzających się czynności.

Przykładowo, można stworzyć jednocześnie kilka różnych systemów sterowania pociskami, izolując oprogramowanie i minimalizując jego zależność od sprzętu konkretnego komputera docelowego. Wykorzystując ALS jako środowisko projektowe, programiści mogą w łatwy sposób współdzielić dowolne, niezależne od sprzętu docelowego oprogramowanie sterowania pociskami. Stosunek między współdzielonymi i pozostałymi modułami źródłowymi jest określany i kontrolowany przez bazę danych środowiska ALS. Raz zaimplementowane funkcje, takie jak obliczanie trajektorii ruchu lub sterowanie lotem, mogą być używane wielokrotnie w każdym z oddzielnych systemów.

Architektura warstwowa

Środowisko ALS jest zaprojektowane w sposób ułatwiający jego przenoszenie z jednej konfiguracji macierzystej na inną. Zgodnie z wymaganiami stawianymi przez Stoneman, ALS posiada architekturę warstwową opartą na macierzystym systemie operacyjnym lub sprzęcie. Otacza system macierzysty tworząc niezależne od maszyny środowisko projektowo-uruchomieniowe, w którym programiści wykorzystują jednakowe narzędzia i języki poleceń, niezależnie od komputera macierzystego.

Dla użytkowników nie ma różnicy, czy ALS wykorzystuje komputer VAX lub duży IBM jako macierzysty. Jest to koncepcja podobna do zastosowanej w systemie operacyjnym Unix. Standardowe jądro KAPSE jest zestawem programów usługowych zapewniających maszynowo niezależne sprzężenie między narzędziami ALS i leżącym na niższym poziomie systemem macierzystym. Narzędzia programowe występujące w systemie ALS są oparte zwykle na wrodzonym systemie operacyjnym.

Jądro KAPSE stanowi warstwę oddzielającą narzędzia od systemu macierzystego. Do przystosowania ALS na inny system macierzysty wystarczy tylko ponownie zaimplementować KAPSE. Sprzężenie między jądrem KAPSE i narzędziami jest zachowane, natomiast właściwości wewnętrzne są zmieniane w celu sprzężenia z nowym systemem operacyjnym. Tym samym, narzędzia są automatycznie przenoszone między różnymi środowiskami macierzystymi systemu ALS.

Specyfikacja Stoneman wymaga, aby środowisko APSE było otwarte (ang. *open-ended*), tzn. rozszerzalne. W środowisku otwartym każdy użytkownik może rozszerzyć zbiór narzędzi bez specjalnej znajomości architektury APSE, a poszczególne organizacje mogą opracowywać narzędzia do zaspokojenia indywidualnych lub lokalnych potrzeb. Zestaw narzędzi może być też rozwijany zgodnie z postępem technologii, zabezpieczając środowisko przed zestarzeniem. Otwartość eliminuje również zależności ekonomiczne i funkcjonalne od dystrybutora środowiska.

Zasady przenośności

Jednym z założeń projektowych systemu ALS było zwiększenie przenośności narzędzi, użytkowników i projektów. Każdy z rodzajów przenośności nakłada różne wymagania na środowisko.

Przenośność narzędzi odnosi się do przemieszczania narzędzi z jednego środowiska macierzystego ALS do innego. Architektura warstwowa ALS i niezależność od środowiska macierzystego w obrębie KAPSE automatycznie czyni zestaw narzędzi przenośnymi. Możliwość przemieszczenia narzędzi między macierzystymi systemami ALS zwiększa liczbę środowisk mogących je wykorzystywać. Eliminuje się przez to projektowanie i zmniejsza koszty tworzenia nowych narzędzi.

Przenośność użytkowników polega na możliwości przechodzenia programistów od jednego projektu lub środowiska do innego. Ponieważ język poleceń systemu ALS i sprzężenie z KAPSE są stałe dla wszystkich środowisk macierzystych ALS, to użytkownik nowej konfiguracji macierzystej widzi środowisko identycznie jak poprzednie, zarówno pod względem funkcjonalności jak i sprzężenia. Minimalizuje się przez to potrzebę szkolenia użytkowników w wypadku zmiany projektu lub miejsca pracy.

Przenośność projektu oznacza zdolność przenoszenia kompletnych projektów między środowiskami macierzystymi. W miarę rozwoju programu od fazy implementacji do pielęgnacji, odpowiedzialność za produkt jest przenoszona z pracowników wytwarzających oprogramowanie na zespół zajmujący się pielęgnacją. Dane opisujące projekt i jego historię, przechowywane w bazie danych ALS, można przenosić do innej bazy danych, gdzie będą dostępne i zrozumiałe dla osób nie będących autorami.

GŁÓWNE SKŁADNIKI

Z perspektywy użytkownika ALS ma trzy główne składniki: język poleceń, bazę danych środowiska i zestaw narzędzi.

Język poleceń wzorowany na Adzie

Język poleceń służący wywoływaniu i kontrolowaniu działania narzędzi jest uniwersalny. Nadaje się do wykorzystywania w trybie interakcyjnym prosto z terminala i w trybie wsadowym przy wykorzystaniu procedur poleceń. Dla zapewnienia zgodności postać leksykalna, struktury składniowe i sterujące języka poleceń wzorowano na Adzie.

Procedury poleceń można pisać używając pętli programowych i instrukcji *IF-THEN-ELSE* Ady. Wykorzystując składnię podobną do wywołań procedur w Adzie – włącznie z przekazywaniem parametrów – procedury poleceń mogą wywoływać się wzajemnie lub wywoływać programy Ady. Język poleceń różni się tym od Ady, że jest ukierunkowany na przetwarzanie napisów i nie ma tak ścisłej typizacji. Jest również prostszy od Ady i przeznaczony do interpretacji, a nie kompilacji.

Baza danych środowiska

Baza danych środowiska, EDB (ang. *environment data base*), zawiera wszystkie informacje generowane i obsługiwane w systemie ALS. EDB stanowi system plików służących do przechowywania wszystkich informacji związanych z projektem. Może służyć do przechowywania dowolnych informacji tekstowych lub dwójkowych, jak np. programy źródłowe, programy wynikowe, dokumentacja, specyfikacje, pliki danych, informacje organizacyjne, opisy testów, zestawy testów i ich wyniki.

EDB dostarcza standardowego środka komunikacji dla narzędzi systemu ALS. Każdy obiekt występujący w bazie danych składa się z informacji i charakterystyk opisujących go i jego powiązania z innymi obiektami bazy danych. Struktura bazy danych przypomina hierarchiczne, katalogowe struktury plikowe większości wielożytkowych systemów operacyjnych, takich jak VMS czy Unix. Podstawowa różnica polega na tym, że baza systemu ALS ma wbudowane funkcje zarządzania konfiguracją.

Konwencje dotyczące zawartości i struktury EDB, w powiązaniu z narzędziami działającymi na niej, dostarczają środków do sprawowania opieki nad oprogramowaniem w całym cyklu istnienia. Dzięki elastycznej strukturze systemu plików dane są podzielone w sposób odzwierciedlający podział pracy w projekcie lub organizację programu.

Narzędzia systemu ALS wspomagają wszystkie osoby związane z projektem, tj. personel kierowniczy, pomocniczy oraz programistów. Dotyczy to również wszystkich poziomów współpracy w obrębie zespołów, począwszy od jego pojedyńczych członków, a skończywszy na całym zespole projektowym. Użytkownicy mogą mieć własne przestrzenie robocze, pracować nad różnymi częściami programu jednocześnie i współdzielić informacje między sobą. Dostęp do określonych informacji i narzędzi może być rozdzielony między poszczególnych użytkowników lub ich grupy.

Dzięki takiej elastyczności wśród członków zespołu realizującego projekt można organizować i koordynować przepływ informacji. Można też rejestrować wiele wersji modułów programu, a pliki całkowicie zabezpieczać przed modyfikacjami i wykorzystywać jako podstawę do odtworzenia poprzednich produktów.

Zestaw narzędzi podstawowych

System ALS zawiera obszerny zestaw narzędzi działających na bazie danych tego środowiska i wspomagających wszystkie fazy opracowywania i pielęgnowania oprogramowania. Narzędzia można sklasyfikować według funkcjonalności włączając generowanie programu, zarządzanie bazą danych, zarządzanie konfiguracją kontrolowanie dostępu i zabezpieczeń, zarządzanie plikami, operacje tekstowe i wyświetlanie. Mimo, że system ALS udostępnia wiele różnych usług, to użytkownik rozpoczynający opracowywanie programu nie musi znać ich wszystkich – system jest łatwy w obsłudze dla początkujących.

Narzędzia wytwarzania programów pomagają w opracowywaniu wykonywalnych programów Ady dla określonego środowiska docelowego. Podstawowy zestaw narzędzi tego rodzaju obejmuje kompilator Ady, asembler importujący kod zewnętrzny, eksporter wytwarzający moduł wykonywalny w określonym środowisku docelowym i konsolidator. Dodatkowe narzędzia wspomagające wytwarzanie programów stanowią symulatory i analizatory wydajności.

Lamport L.: Uproszczone specyfikowanie systemów współbieżnych (3)
INFORMATYKA 1989,nr 1, s. 2
Trzecia część prezentacji nowej metody ułatwiającej specyfikowanie systemów współbieżnych.

Lamport L.: Simplified concurrent systems specifying (3)
INFORMATYKA 1989, No. 1, p. 2
Third part of presentation of a new method, which makes concurrent systems specifying easy.

Lamport L.: Vereinfachte Spezifizierung der parallelen Systeme (3)
INFORMATYKA 1989, Nr. 1, S. 2
Dritter Teil einer Präsentation von neuer Methode, die Spezifizierung der parallelen Systeme erleichtert.

Sysło M. M.: Maszyny i algorytmy równoległe (3)
INFORMATYKA 1989, nr 1, s. 6
Trzecia część przeglądu zrealizowanych modeli równoległych systemów komputerowych oraz zaprojektowanych dla nich algorytmów, zawierająca charakterystykę maszyn i algorytmów systolicznych.

Sysło M. M: Parallel machines and algorithms (3)
INFORMATYKA 1989, No. 1, p. 6
Third part of the survey presenting realized models of parallel computer systems and for them designed algorithms, which includes characteristics of systolic machines and algorithms.

Sysło M. M.: Parallele Maschinen und Algorithmen (3)
INFORMATYKA 1989, Nr. 1, S. 6
Dritter Teil einer Übersicht von realisierten Modellen der parallen Rechnersysteme, der eine Charakteristik von systolischen Maschinen und Algorithmen enthält.

Kłopotek M. A.: TMK – pakiet syntezy mowy dla IBM PC
INFORMATYKA 1989, nr 1, s. 10
Charakterystyka problemu komputerowej syntezy mowy oraz rozwiązań pakietu TMK do głosowej transformacji tekstów w języku polskim.

Kłopotek M. A.: TMK – a package of speach synthesis for IBM PC
INFORMATYKA 1989, No. 1, p. 10
Characteristics of the problem of speach synthesis by computer, as well as of solutions of the TMK package for voice transforming of polish language text.

Kłopotek M. A.: TMK – ein Sprachsynthesepaket für IBM PC
INFORMATYKA 1989, Nr. 1, S. 10
Eine Charakteristik des Problems von Sprachsynthese mit Hilfe eines Computers und Lösungen des TMK-Pakets für Stimmeumwandlung der Texte in polnischer Sprache.

Struk Z., Zabża-Tarka E.: Prosty system zarządzania bibliograficzną bazą danych dla mikrokomputera IBM PC/XT
INFORMATYKA 1989, nr 1, s. 13
Charakterystyka funkcjonalnych i programowych rozwiązań modularnego systemu zarządzania bibliograficzną bazą danych, dostosowanego do możliwości mikrokomputera typu IBM PC/XT.

Struk Z., Zabża-Tarka E.: A simple bibliographic data base managament system for IBM PC/XT microcomputer
INFORMATYKA 1989, No. 1, p. 13
Charakteristics of functional and software solutions of the moduler bibliographic data base management system, adjusted to possibilities of the IBM PC/XT microcomputer.

Struk Z., Zabża-Tarka E.: Einfacher bibliographischer Datenbankverwaltungssystem für IBM PC/XT Mikrocomputer
INFORMATYKA 1989, Nr. 1, S. 13
Eine Charakteristik der Funktional – und Programmlösungen von modulareñ und zu Möglichkeiten des IBM PC/XT angepassten Verwaltungssystem der bibliographischen Datenbank.

Pawlak B.: MIC – metaasembler mikroprogramów
INFORMATYKA 1989, nr 1, s. 15
Charakterystyka zasad działania metaasemblerów mikroprogramów oraz rozwiązań metaasemblera MIC, opracowanego w Instytucie Informatyki Politechniki Warszawskiej.

Pawlak B.: MIC – a microprogram metaassembler
INFORMATYKA 1989, No. 1, p. 15
Characteristics of microprogram metaassembler operation principles and solutions of the MIC metaassembler, which was elaborated in the Informatics Institute of Warsaw Technical University.

Pawlak B.: MIC – ein Metaassembler für Mikroprogramme
INFORMATYKA 1989, Nr. 1, S. 15
Eine Charakteristik von Operationsgrundsätzen der Metaassembler für Mikroprogramme und von Lösungen des MIC-Metaassemblers, der im Institut für Informatik der Warschauer Technischen Universität erarbeitet wurde.

Tucholski A.: NETBIOS – zasada działania i sposób użytkowania (2)
INFORMATYKA 1989, nr 1, s. 20
Druga część charakterystyki standardowego zbioru procedur do sprzęgania adaptera sieci komputerowej z systemem operacyjnym DOS, zawierająca przykłady użycia tych procedur.

Tucholski A.: NETBIOS – operation principle and application method (2)
INFORMATYKA 1989, No. 1, p. 20
Second part of characteristics of standard procedure set for interfacing computer network adapter with DOS operating system, which includes examples for such procedures application.

Tucholski A.: NETBIOS – Operationsgrundsatz und Anwendungsweise (2)
INFORMATYKA 1989, Nr. 1, S. 20
Zweiter Teil einer Charakteristik von standarter Prozedurmenge zum Koppeln des Netzadapters mit dem DOS-Betriebssystem, der einige Anwendungsbeispiele von solchen Prozeduren umfasst.

Gizdoń H., Pawlak A., Wrona W.: Język opisu sprzętu VHDL – podstawowe mechanizmy (2)
INFORMATYKA 1989, nr 1, s. 23
Druga część szczegółowej charakterystyki języka opisu sprzętu VHDL, zawierająca omówienie rozwiązania dalszych mechanizmów wersji 7.2 tego języka.

Gizdoń H., Pawlak A., Wrona W.: VHDL hardware description language – basic mechanisms (2)
INFORMATYKA 1989, No. 1, p. 23
Second part of detailed characteristics of the VHDL hardware description language, which includes discussion of further solutions in the 7.2 version of this language.

Gizdoń H., Pawlak A., Wrona W.: VHDL-Hardwarebeschreibungssprache – die Grundmechanismen (2)
INFORMATYKA 1989, Nr. 1, S. 23
Zweiter Teil einer detailierten Charakteristik von VHDL-Hardwarebeschreibungssprache, der eine Besprechung von Lösungen der weiteren Mechanismen in Version 7.2 dieser Sprache anthält.

Bielecki J.: Turbo C. Wywoływanie funkcji, operacje na rejestrach, wstawki asemblerowe
INFORMATYKA 1989, nr 1, s. 26
Kontynuacja praktycznych wskazówek i przykładów stosowania niektórych specyficznych rozwiązań języka programowania Turbo C na mikrokomputerach klasy IBM PC.

Bielecki J.: Turbo C. Functions calling, register operations, assembler inserts
INFORMATYKA 1989, No. 1, p. 26
Continuation of practical advices and examples for application of some specific solutions of the Turbo C language on microcomputers of IBM PC class.

Bielecki J.: Turbo C. Funktionsaufrufe, Registeroperationen, Assemblereinlagen
INFORMATYKA 1989, Nr. 1, S. 26
Fortsetzung von praktischen Hinweisen und Beispielen über spezifische Lösungen der Turbo C-Programmierungssprache in Anwendung auf Mikrorechner der IBM PC-Klasse.

Panie Redaktorze,

Z pewnym opóźnieniem przesyłam mały komentarz do terminologii (sprawy są nadal aktualne) à propos tekstu podpisanego WMT w numerze 2/1988 **INFORMATYKI** (przedruk z Biuletynu PTI).

Tekst zawiera takie uwagi, którym można tylko przyklasnąć, ale także takie, z którymi zgodzić się nie sposób (jest w nim również – niepotrzebnie – nutka jakby zacietrzewienia).

Niezmiernie trafne, na przykład, są uwagi o czasowniku angielskim **to control** i jego nieprawidłowych tłumaczeniach w polszczyźnie. Natomiast z uwagami o przymiotniku **kompatybilny** nie zgodziłbym się. Pierwsze znaczenie słowa *zgodny* to tyle co *niekłótliwy, chętny do współdziałania*. Dopiero drugi sens to *wiernie odpowiadający czemuś* (np. w wyrażeniu *za zgodność z oryginałem*). Słowo *kompatybilny* ma znaczenie specyficzne: *sprzętowo zgodny* (w dziedzinie komputerów). Tylko jedno słowo więcej a przynosi dodatkową precyzję. Nie oddaje tej precyzji wcale słowo *zgodny* i wymagałoby użycia jeszcze dodatkowych słów, aby dorównać terminowi *kompatybilny*.

Propozycja zastąpienia **interfejsu** (czy **interface'u**) *pośrednictwem* to kompletne nieporozumienie. *Pośrednictwo* to po pierwsze pojęcie abstrakcyjne, po drugie oznaczające stan i teraz nagle miałoby oznaczać bardzo konkretne urządzenie techniczne?! Zatem powstałby nowy homonim, którego znaczenia w dodatku dotyczyłyby kompletnie różnych dziedzin rzeczywistości w jej ujmowaniu językowym. Tymczasem homonimy, jak wiemy, nie przyczyniają się do większej jasności języka. Lepszy już byłby *interfejs* czy w ostateczności *łącznia* – jak chcą niektórzy.

Co do **joysticka**, ktoś kiedyś powiedział *grajrączka* i uważam, że słowo to dobrze brzmi i oddaje funkcję przedmiotu. Aczkolwiek, trzeba przyznać, polszczyzna nie lubi złożeń słownych, jak np. już zakorzeniony *światopogląd*, czy okropna *kursokonferencja*. *Orczykowi* jestem przeciwny, ponieważ ma inne znaczenie i wywołuje inne skojarzenia – nie mówiąc o tym, że byłby to znów homonim.

Dalej, jeśli chodzi o uwagę autora, iż polszczyzna nie przyjmuje słów angielskich (oprócz dżemu). Jeszcze jak przyjmuje! Zacytuję tylko niektóre spośród wielu słów angielskich w polszczyźnie: sweter, pulower, boks, tenis, kort, net, gol, korner, forhend, smecz, aut, brydż, poker, off-side, walkower, outsider, fair play, dżentelmen, overlock, inlet czyli wsypa (do poduszki), establishment, lord, buldog, caravaning, camping, trawler, autobus, trolejbus, booking, banknot, jeep, skuter, szyper, trymerka, dżinsy, krakersy, dżersej, hot-dogi (przepraszam, że nie zadbałem w tym wyliczeniu o chronologię przejmowania owych słów).

Wypada tylko dodać, że niegdyś wpływy angielskiego były niezmiernie małe w porównaniu do łaciny i niemieckiego, a teraz są znaczne.

Z pozdrowieniami

A. KAMIEŃSKI

CENTRALNY ZWIĄZEK SPÓŁDZIELNI INWALIDÓW
CENTRALNY OŚRODEK KOORDYNACJI SZKOLENIA
Centrum Kształcenia Inwalidów im. Dr HANNY DWORAKOWSKIEJ

05-510 Konstancin-Jeziorna ul. Gąsiorowskiego 12/14

INFORMACJA

o naborze kandydatów do Centrum Kształcenia Inwalidów na KURS OBSŁUGI KOMPUTERA

Do Centrum przyjmowane są osoby ze schorzeniami narządu ruchu, przede wszystkim inwalidzi z uszkodzeniami rdzenia kręgowego, amputacją dolnych kończyn, inni inwalidzi narządu ruchu, poruszający się za pomocą wózka inwalidzkiego.
Wiek kandydatów: 18–35 lat.
Od kandydatów wymagana jest:
- ukończona co najmniej szkoła średnia
- zachowana pełna sprawność kończyn górnych
- samodzielność w obsługiwaniu się i poruszaniu.

Przyjęcie do Centrum wykluczają: odleżyny, choroby psychiczne, niedorozwój lub ociężałość umysłowa, choroby z dynamizmem pogarszania, choroby upośledzające sprawność umysłową lub manualną (kończyn górnych), epilepsja, hemofilia, zaawansowane wady wzroku, słuchu, serca.

Podczas pobytu w Centrum słuchaczom zapewnia się:
- zakwaterowanie w internacie
- wyżywienie (odpłatnie)
- doraźną i specjalistyczną opiekę medyczną
- zabiegi terapeutyczne
- pomoc administracyjno-prawną w załatwianiu spraw socjalnych
- możliwość zdobycia prawa jazdy kat. „B"
- uczestnictwo w korzystaniu z dóbr kultury (kino, teatr, wycieczki), pogłębianie zainteresowań w ramach zajęć pozalekcyjnych.

Kandydaci do Centrum powinni złożyć następujące dokumenty:
1. Podanie i życiorys
2. Orzeczenie KIZ o przyznanej grupie inwalidzkiej
3. Świadectwo szkolne
4. Wywiad socjalny
5. Zaświadczenie lekarskie, stwierdzające rozpoznanie schorzenia, aktualny stan zdrowia, wydolność rąk
6. Charakterystykę psychologiczną
7. Odcinek renty lub zaświadczenie o zarobkach rodziców
8. Zaświadczenie o możliwości zatrudnienia po ukończeniu kursu
9. 2 fotografie (aktualne)

Szczegółowe informacje można uzyskać pisemnie lub telefonicznie, tel. 56-45-57, 56-32-40, 56-32-60.
Dokumenty prosimy kierować pod adresem:
Centrum Kształcenia Inwalidów im. Dr Hanny Dworakowskiej 05–510 Konstancin–Jeziorna, ul. Gąsiorowskiego 12/14 woj. stołeczne warszawskie.

EO/1291/88

Warunki prenumeraty w roku 1989

Prenumeratorzy zbiorowi – jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat wyłącznie na blankiecie „wpłata-zamówienie" (jest to „polecenie przelewu" rozszerzone dla potrzeb Wydawnictwa o część dotyczącą zamówienia).

Blankiety te będą dostarczane dotychczasowym prenumeratom przez Zakład Kolportażu. Nowi prenumeratorzy otrzymają je po zgłoszeniu zapotrzebowania (pisemne lub telefoniczne) w Zakładzie Kolportażu.

Prenumeratorzy indywidualni – osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty. Wpłacać należy na konto: Państwowy Bank Kredytowy III/O Warszawa nr 370015-7490-139-11.

Prenumerata ulgowa – przysługuje wyłącznie osobom fizycznym – członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły. Sposób zamawiania prenumeraty ulgowej jest taki sam jak prenumeraty indywidualnej. W prenumeracie ulgowej można zamówić tylko po jednym egzemplarzu każdego czasopisma.

Uwaga! Miesięcznik „Aura" może być zamawiany w prenumeracie ulgowej również przez uczniów szkół ogólnokształcących.

Prenumerata ze zleceniem wysyłki za granicę – zamawia się tak, jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy.

Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Wpłaty na prenumeratę przyjmowane są w terminach:

- do 10 listopada na każdy kwartał, I i II półrocze oraz cały rok następny,
- do 28 lutego na II, III i IV kwartał oraz II półrocze,
- do 31 maja na III i IV kwartał oraz II półrocze,
- do 31 sierpnia na IV kwartał.

Zmiany w prenumeracie można zgłaszać pisemnie tylko w wyżej wymienionych terminach.

Informacji o prenumeracie udziela – Zakład Kolportażu Wydawnictwa NOT SIGMA (ul. Bartycka 20, 00–716 Warszawa), skr. poczt. 1004, 00–950 Warszawa, tel. 40-00-21 wew. 248, 249, 293, 297, 299 lub 40-30-86 i 40-35-89.

Egzemplarze archiwalne czasopism – można nabyć za gotówkę w Klubie Prasy Technicznej, Warszawa ul. Mazowiecka 12 (tel. 26-80-16), lub zamówić pisemnie. Zamówienia na egzemplarze archiwalne czasopism przyjmuje: Zakład Kolportażu, Dział Handlowy, 00–950 Warszawa, skr. poczt. 1004 (tel. 40-37-31), na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

miesięczna		kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
300 zł	60 zł	900 zł	180 zł	1800 zł	360 zł	3600 zł	720 zł