ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1984

Seria: INFORMATYKA z. 7 Nr kol. 809

Krzysztof NAŁĘCKI

Instytut Informatyki Czasu Rzeczywistego
Politechniki Śląskiej

WSPOMAGANIE PRODUKCJI OPROGRAMOWANIA MIKROPROCESORÓW - KONCEPCJA USOM

Streszczenie. W artykule przedstawiono ogólną charakterystykę problemów produkcji oprogramowania, w szczególności dla mikroprocesorów oraz własności stosowanego oprogramowania narzędziowego. Omówiono używanie różnych języków programowania i różnych typów mikroprocesorów. Zaproponowano strukturę systemu wspomagającego produkcję oprogramowania dla mikroprocesorów. Scharakteryzowano zakres implementacji proponowanej koncepcji w postaci USOM - Uniwersalnego Systemu Oprogramowania Mikroprocesorów[1].

1. WSTĘP

Przez produkcję oprogramowania rozumieć będziemy wytwarzanie oprogramowania z użyciem odpowiednich narzędzi. Specyfika mikroprocesorów w tym zakresie uzewnętrznia się zwykle brakiem odpowiedniego oprogramowania narzędziowego lub koniecznością stosowania specjalizowanych stanowisk uruchomieniowych o rozbudowanej konfiguracji sprzętowej. Rozbudowa konfiguracji jest wymuszona właściwościami eksploatacyjnymi oprogramowania narzędziowego. Wygodny w użyciu edytor tekstu źródłowego wymaga pamięci dyskowej. Kompilator języka wysokiego poziomu może funkcjonować dopiero przy odpowiednio dużym obszarze pamięci operacyjnej. Administracja modułami oprogramowania użytkowego w trakcie wytwarzania wymaga systemu operacyjnego ze sprawnym systemem plików. Tworzenie, przechowywanie i wykorzystanie bibliotek procedur możliwe jest przy użyciu pamięci dyskowej.

Użycie stanowiska uruchomieniowego jest konieczne w fazie scalania oprogramowania i sprzętu systemu mikroprocesorowego. Ponadto stanowiska uruchomieniowe wykonywane są z reguły jako systemy z jednym użytkownikiem. Zatem, jeśli stanowisko uruchomieniowe używane będzie zarówno do produkcji oprogramowania, jak i do uruchamiania sprzętu, stanie się potencjalnie wąskim gardłem, szczególnie w przypadku większych projektów. Powiela-

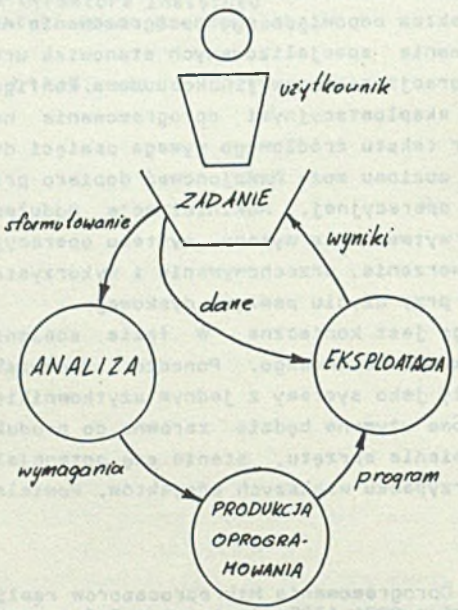
[1] Projekt USOM - Uniwersalny System Oprogramowania Mikroprocesorów realizowany jest od 1981 r. w Laboratorium ODRA 1305 Instytutu Informatyki Czasu Rzeczywistego Politechniki Śląskiej w ramach problemu węzłowego 06.4.

nie stanowisk uruchomieniowych jest ekonomicznie zbyt kosztowne. Proponowanym rozwiązaniem jest wykorzystanie wielodostępnego systemu komputerowego ogólnego przeznaczenia do produkcji oprogramowania mikroprocesorów. Dodatkową zaletą takiego rozwiązania, oprócz zwiększenia liczby równocześnie dostępnych stanowisk pracy dla programistów, jest fakt na ogół dobrego wyposażenia dużego systemu komputerowego w narzędzia programowe, usprawniające pracę programisty i duża różnorodność urządzeń peryferyjnych. Ten sposób produkcji oprogramowania nazywać będziemy skrośnym, gdyż komputer, dla którego produkuje się oprogramowanie (komputer docelowy) różni się od komputera, przy użyciu którego to oprogramowanie jest produkowane (komputera macierzystego). Przykładami skrośnych systemów produkcji oprogramowania mikroprocesorów są: UMDS-BSO [1], MicroSim-DMEE [2], PasPort-Intermetrics [3]. Do klasy tej należy również USOM [4, 5, 6].

W dalszej części artykułu przedstawiona zostanie charakterystyka procesu produkcji oprogramowania i wynikające stąd właściwości oprogramowania narzędziowego oraz ogólna koncepcja uniwersalnego systemu wspomagającego oprogramowanie mikroprocesorów i jej realizacja w postaci USOM.

2. PRODUKCJA OPROGRAMOWANIA

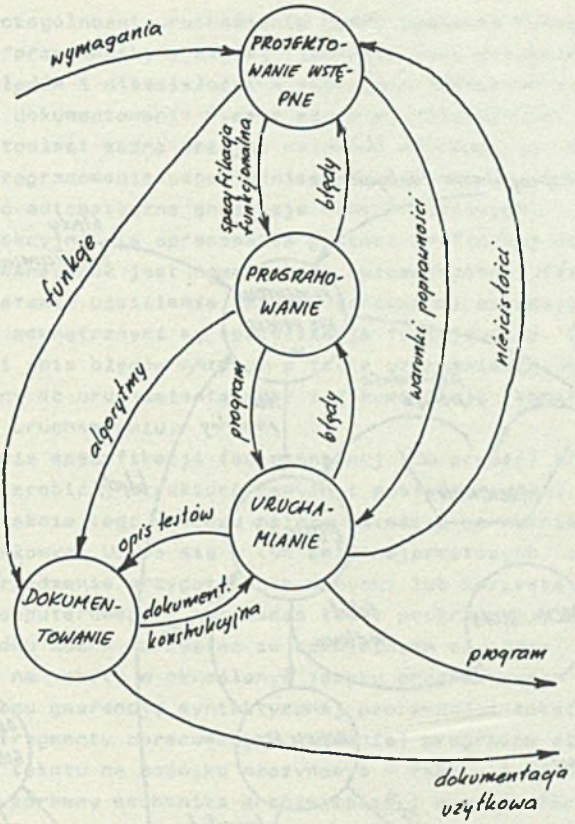
Produkcja oprogramowania jest działalnością ukierunkowaną na uzyskanie programu komputerowego, którego eksploatacja umożliwi efektywne rozwiązanie pewnego zadania lub klasy zadań (rys. 1). Celem analizy jest przekształcenie sformułowania treści zadania w wymagania dla oprogramowania.



Rys. 1. Miejsce produkcji oprogramowania w cyklu życia programu

W procesie produkcji oprogramowania wyróżnimy następujące etapy składowe (rys. 2):

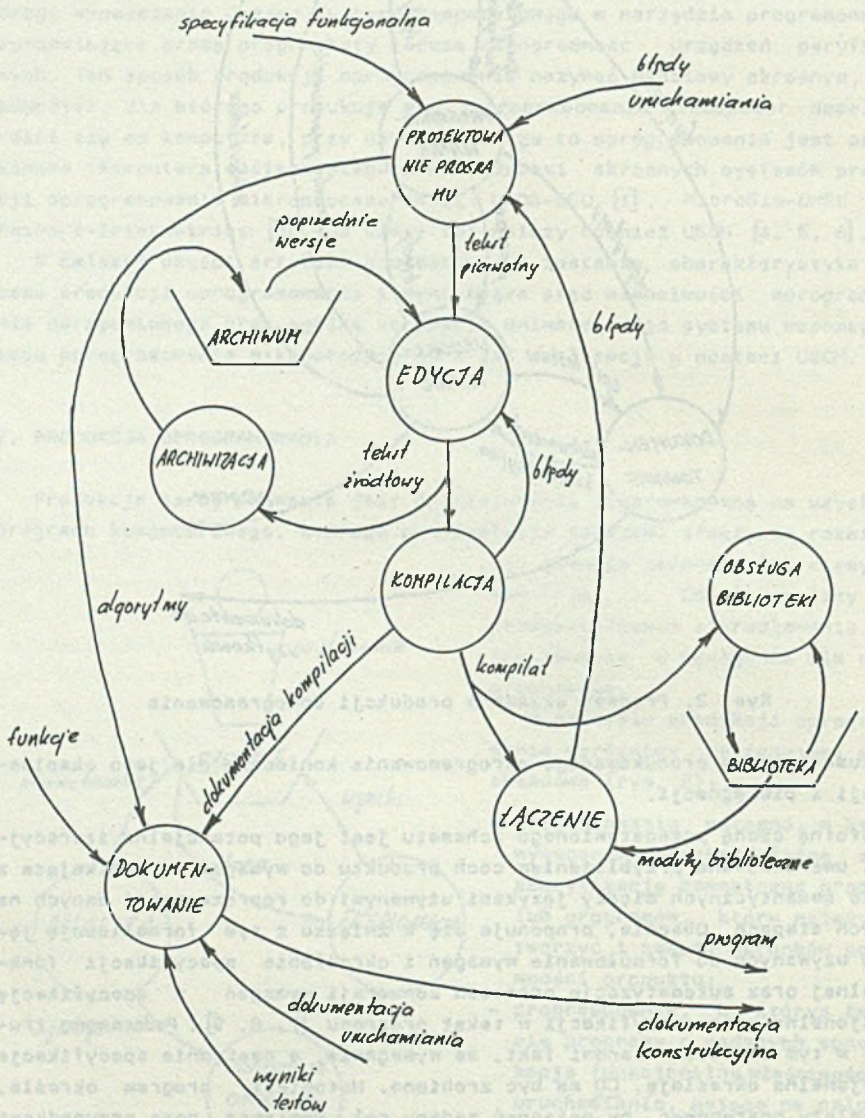
- projektowanie wstępne, w którym wymagania przekształcane są w specyfikację zewnętrzną programu lub programów, które należy wytworzyć i zespół warunków poprawności produktu;
- programowanie, w którym tworzy się programy o zadanych specyfikację funkcjonalną właściwościach;
- uruchamianie, mające na celu weryfikację poprawności wytworzonego programu w aspekcie postawionych wymagań;



Rys. 2. Procesy składowe produkcji oprogramowania

- dokumentowanie produkowanego oprogramowania konieczne dla jego eksploatacji i pielęgnacji.

Istotną cechą przedstawionego schematu jest jego potencjalna iteracyjność, uwarunkowana przybliżeniem cech produktu do wymagań, a wynikające z różnic semantycznych między językami używanymi do reprezentacji danych na różnych etapach. Obecnie, proponuje się w związku z tym formalizację języków używanych do formułowania wymagań i określania specyfikacji funkcjonalnej oraz automatyzację procesów konwersji wymagań w specyfikację funkcjonalną oraz specyfikacji w tekst programu [7, 8, 9]. Podstawową trudność w tym zakresie stanowi fakt, że wymaganie, a następnie specyfikacja funkcjonalna określają, CO ma być zrobione. Natomiast program określa, JAK należy postępować, by osiągnąć zadany cel. Ponieważ, poza przypadkami trywialnymi, ten sam efekt można osiągnąć różnymi sposobami (i różnym kosztem), pełna automatyzacja produkcji oprogramowania jest niemożliwa i nie-



Rys. 3. Przebieg programowania

celowa. W szczególności uruchamianie oprogramowania wymaga obecności "intruza" albo "przeciwnika", którego zadaniem jest wykrycie maksymalnie dużej liczby błędów i nieścisłości w reakcjach (wynikach wykonania) programów. Również dokumentowanie wymaga aktywnej działalności człowieka. Bez wątpienia natomiast można przyjąć celowość wspomagania wszystkich etapów produkcji oprogramowania odpowiednimi narzędziami programowymi. Przykładami mogą być automatyczna generacja danych testowych, różnego rodzaju programy redakcyjne dla opracowania postaci graficznej dokumentacji. Najbardziej zaawansowana jest powszechnie automatyzacja fazy programowania. Rys. 3 przedstawia uściślanie działań (procesów) składających się na tę fazę. Danymi zewnętrznymi są specyfikacja funkcjonalna (nazywana również zewnętrzną) i opis błędów wykrytych w fazie uruchamiania. Wynikiem jest program kierowany do uruchamiania wraz z dokumentacją konstrukcyjną, wykorzystywaną w uruchamianiu.

Na podstawie specyfikacji funkcjonalnej (CO zrobić) projektuje się algorytmy (JAK zrobić), strukturę danych i postać programu. Tekst programu uzyskany w efekcie tego procesu należy umieścić na nośniku maszynowym np. w zbiorze dyskowym. Używa się w tym celu najprostszych środków technicznych (tzw. urządzenia przygotowania danych) lub korzysta z interakcyjności systemu komputerowego i wprowadza tekst programu w ramach dialogu. W tym przypadku można korzystać ze specjalnych edytorów syntaktycznie orientowanych na teksty w określonym języku programowania [10]. Uzyskuje się dzięki temu gwarancję syntaktycznej poprawności tekstu. Czasem można wykorzystać fragmenty opracowanych wcześniej programów dla skrócenia czasu tworzenia tekstu na nośniku maszynowym - tekstu źródłowego. W tym celu musi istnieć sprawny mechanizm archiwizacji i administracji modułami źródłowymi. Przykładem może być wykorzystanie opisu struktury zbioru danych, przetwarzanego w kilku programach. Uzyskany w procesie edycji tekst źródłowy jest przekazywany do kompilacji. Powinien zostać również zarchiwizowany dla umożliwienia wprowadzania poprawek. Kompilacja i edycja powtarzane są aż do uzyskania bezbłędnych wyników. Raport kompilacji staje się elementem dokumentacji, a uzyskany zbiór modułów wynikowych (kompilat) należy uzupełnić w procesie łączenia o moduły biblioteczne, wcześniej opracowane, dla uzyskania kompletnego programu. Moduł wynikowy może zostać również skierowany do biblioteki do wykorzystania później (po opracowaniu pozostałych modułów produkowanego programu lub w innych programach). Biblioteka wymaga odpowiednich mechanizmów obsługi, umożliwiających wprowadzenie nowych modułów, zastępowanie i/lub usuwanie błędnych bądź nieprzydatnych oraz udostępnianie potrzebnych w procesie łączenia.

Biblioteka modułów wynikowych łącznie z archiwum tekstów źródłowych i dokumentacją modułów tworzy bazę danych, nad którą realizuje się produkcję oprogramowania.

3. ŚRODKI WSPOMAGANIA OPROGRAMOWANIA

Wspomaganie programowania i ogólniej produkcji oprogramowania ma na celu obniżenie kosztów procesu, głównie przez skrócenie czasu realizacji oraz podniesienie jakości produktu (zmniejszenie liczby błędów, lepszą dokumentację). Środki do tego celu używane - oprogramowanie narzędziowe - nazywa się również środowiskiem wspomagającym programowanie (Programming Support Environment-PSE) [11]. Środowisko wspomagające programowanie operuje na modułach, tzn. na wyodrębnionych i rozróżnialnych zestawach informacji. Moduł opisany jest zbiorem atrybutów. Przykładami atrybutów mogę być: nazwa, typ, wskaźnik wersji, kontrola dostępu. Pewne atrybuty ustala i nadaje im wartość użytkownik, inne pozostają w administracji programów narzędziowych.

Wyprowadzeniem modułu będziemy nazywać proces transformacji pewnego zbioru modułów przy użyciu odpowiednich narzędzi w celu uzyskania modułu (lub modułów) bliższych wykonaniu. Przykładami wyprowadzania są: kompilacja i łączenie (rys. 3). Moduł, który nie może zostać wyprowadzony przy użyciu dostępnych narzędzi nazywać będziemy modułem abstrakcyjnym lub pierwotnym. Moduł taki musi być utworzony przez użytkownika, w przeciwieństwie do modułów wyprowadzalnych, tworzonych i obsługiwanych przez programy narzędziowe.

Dla modułu wyprowadzonego istotnym zespołem atrybutów jest opis historii wyprowadzenia, specyfikujący moduły źródłowe i narzędzia użyte do jego utworzenia.

Rewizją nazywać będziemy proces transformacji pewnego modułu w celu uzyskania zastępnika.

Typowym przykładem rewizji jest edycja. W wyniku rewizji otrzymujemy nową wersję modułu. Rewizję stosować można również do modułu abstrakcyjnego. Rozróżnialność różnych wersji zapewnia atrybut nazwany wskaźnikiem wersji. W najprostszym przypadku może to być numer wersji, zwiększany w kolejnych rewizjach. Inna możliwość - rejestracja daty rewizji - daty narodzin modułu wraz z datą narodzin wersji poddawanej rewizji. W tym przypadku przestaje obowiązywać liniowe uporządkowanie zbioru wersji, co może być korzystne przy tworzeniu wielu wariantów wersji "próbnych".

Znajomość historii wyprowadzenia pewnego modułu pozwala na automatyczne powtórzenie procesu wyprowadzania przy użyciu znowelizowanych wersji modułów źródłowych jak i aktualnych wersji programów narzędziowych, poczynając od poziomu modułów abstrakcyjnych.

Ważne znaczenie praktyczne dla użytkownika może mieć możliwość reversji wyprowadzenia lub rewizji. Najprostszym rozwiązaniem jest przechowywanie (archiwizacja) wszystkich wersji modułów abstrakcyjnych i modułów z nich wyprowadzonych oraz wersji stosowanych programów narzędziowych. Rozwiązanie to powoduje jednak znaczną zajętość pamięci masowej i bez sprawnych mechanizmów administrowania zbiorami traci sens. Rozwiązanie wystar-

czające to przechowywanie modułów abstrakcyjnych i historii wyprowadzania. Zapewnia się w ten sposób możliwość ponownego przeprowadzania procesu wyprowadzania. Porównanie kosztów powtórzenia wyprowadzenia (lub pewnego jego fragmentu) z kosztem przechowywania i administrowania wielu wersji wyprowadzalnych powinno wskazać wybór właściwego wariantu postępowania.

Zestaw środków wspomagających programowanie charakteryzują jego własności funkcjonalne. Niektóre z nich omówimy.

A. Poziom językowy - liczba, poziom i stopień integracji języków używanych do definiowania modułów abstrakcyjnych.

B. Zakres konfiguracji docelowych - zwykle konfiguracja systemu komputerowego używanego w produkcji oprogramowania jest znacznie bardziej rozbudowana niż konfiguracja sprzętu, dla którego oprogramowanie się produkuje. Dotyczy to w szczególności systemów mikroprocesorowych. Dlatego istotną cechą oprogramowania wspomagającego jest zakres jego uniwersalności, rozumianej jako różnorodność konfiguracji docelowych oraz otwartość na potencjalne zmiany i rozwój, spowodowane postępem technicznym.

C. Komunikacja z użytkownikiem, przez którą rozumieć będziemy:

- wymagany stopień znajomości użytkowania bazowego systemu operacyjnego i/lub systemu zbiorów;
- możliwość współbieżnego wykorzystywania przez wielu użytkowników;
- możliwość stosowania urządzeń końcowych różnych typów, w tym konwersacyjnych i wsadowych;
- możliwość automatyzowania na poziomie języka zleceń często używanych sekwencji i mechanizm wartości domniemanych;
- stopień ochrony przed błędami użytkownika, diagnostyka błędów i możliwość informowania o sposobie poprawnego użytkowania.

D. Stopień scalenia i unifikacji narzędzi - programy narzędziowe są zwykle tworzone w pewnym stopniu niezależnie. Oczywisty fakt wykorzystywania ich wspólnie w ramach zespołu tworzącego środowisko wspomagania programowania uwypukla wszystkie niejednorodności i różnice, co w efekcie może stwarzać użytkownikowi kłopoty w eksploatacji.

E. Precyzja narzędzi - tańsze i efektywniejsze w eksploatacji oraz bardziej niezawodne są niewielkie specjalizowane programy narzędziowe w przeciwieństwie do uniwersalnych o wielu funkcjach. Niewielkie ("precyzyjne") programy narzędziowe pozwalają na bardziej "drobnoziarnistą" obróbkę modułów, zwiększając elastyczność działań użytkownika. Łatwiej również osiągnąć otwartość na zmiany. Wymagany jest za to rozbudowany mechanizm administrowania narzędziami, ułatwiający kompletowanie aktualnie wymaganego, np. dla realizacji wyprowadzenia, zestawu programów narzędziowych.

F. Wspomagania dokumentowania jako ułatwienia w tworzeniu na bieżąco aktualnej dokumentacji projektu, w dogodnej formie graficznej oraz udostępnianie selektywnie fragmentów dokumentacji.

G. Rejestracja działań dla weryfikacji postępu prac, oceny przydatności i stopnia wykorzystania poszczególnych funkcji, typowych błędów, najczęściej realizowanych sekwencji działań itp. Na tej podstawie można stopniowo ulepszać właściwości eksploatacyjne programów narzędziowych, eliminować narzędzia "niebezpieczne" i przede wszystkim kontrolować przebieg prac produkcyjnych pod względem czasu, kosztów i zaawansowania.

4. JĘZYKI PROGRAMOWANIA

Język programowania jest środkiem do tworzenia modułów abstrakcyjnych. Z tego powodu korzystne są języki wysokiego poziomu, pozwalające na stosowanie odpowiednio wysokiego poziomu abstrakcji przy definiowaniu funkcji modułu i jego struktur danych. Równocześnie wyprowadzenie modułu wykonywalnego w tym przypadku jest na tyle złożone, że praktycznie użytkownik traci możliwość kontroli funkcjonowania sprzętu w trakcie wykonywania modułu. Nie zawsze można się na takie rozwiązanie zgodzić, w szczególności w przypadku systemów bazujących na mikroprocesorach. Wynika stąd konieczność używania różnych języków programowania, o różnym poziomie abstrakcji.

Najniższym poziomem jest poziom języka symbolicznego - asemblera. Zapewnia on pełną kontrolę funkcjonowania sprzętu za cenę konieczności używania wyłącznie operacji elementarnych i fizycznego poziomu organizacji danych. Dla podwyższenia poziomu abstrakcji i przyspieszenia na tej drodze procesu programowania używa się w tym przypadku mechanizmu makroinstrukcji.

W systemie wspomagania programowania, zorientowanym na wiele różnych typów mikroprocesorów występować muszą kompilatory języków symbolicznych tych mikroprocesorów - asemblery. Oficjalną definicję języka symbolicznego mikroprocesora ustala zwykle producent, wprowadzając na rynek asembler wraz z produktem - minimalny warunek wzbudzenia zainteresowania zastosowaniami. Różnice między językami symbolicznymi różnych mikroprocesorów przedstawić można na następujących 4 poziomach:

a) różnice w architekturze mikroprocesorów powodują różnice w zestawie operacji elementarnych - listach rozkazów tych mikroprocesorów;

b) stosuje się różniące się wzajemnie sposoby zapisu żądania wykonania operacji elementarnej - formaty instrukcji języka symbolicznego, różniące się kodami mnemonicznymi rozkazów, postacią kodowania argumentów, stosowanymi separatorami i symbolami organizacyjnymi (np. komentarz);

c) używa się różnych sposobów sterowania procesem kompilacji (dyrektywy) oraz struktury modułu kompilowanego;
d) stosuje się różne formaty zapisu treści modułu ładowalnego.

Z wyżej przedstawionych różnic, technicznie uzasadnione są jedynie wymienione w punkcie a.

Natomiast algorytmy kompilacji, niezależnie od formy języka symbolicznego, są z wyjątkiem różnic powodowanych różną postacią dyrektyw, identyczne. Ponadto przyjęcie koncepcji modułów odsuwa sprawę różnic wymienionych w punkcie d do fazy łączenia modułów.

Zatem można zaproponować częściową unifikację języków symbolicznych różnych mikroprocesorów przez ujednolicenie formatów instrukcji, zestawu dyrektyw i struktury modułów. Dzięki temu można, w stosunkowo prosty sposób, uzupełniać standardowy "szkielet" asemblera o specyficzne dla danego typu mikroprocesorów właściwości zdefiniowane listę rozkazów [12]. Akceptowany przez tak uzyskany asembler język symboliczny będzie się różnił od definicji producenta. Stosując jednak odpowiednio zdefiniowane makroinstrukcje można, przy sprawnym mechanizmie makrogeneracji asemblera, osiągnąć zgodność z językiem producenta dla zapewnienia przenośności modułów źródłowych w języku symbolicznym. Wyodrębnienie fazy łączenia (rys. 3) implikuje kompilację jednoprzebiegową oraz konieczność stworzenia narzędzia dla łączenia modułów - programu łączącego. Jeśli ustali się wspólną postać zapisu modułów skompilowanych, pewien język pośredni, można opracować uniwersalny program łączący. W program ten wbudowywana będzie specjalizowana procedura zapisu treści modułu ładowalnego w formacie narzuconym przez producenta sprzętu. Ujednolicenie formy zapisu modułów skompilowanych pozwala ponadto na opracowanie uniwersalnego zestawu programów obsługi biblioteki modułów, niezależnego od typu mikroprocesora.

Warto zauważyć, że w wyniku pracy asemblera uzyskujemy kompozycję dwóch poziomów abstrakcji. W module wynikowym występują kody operacji, zgodnie z listę rozkazów mikroprocesora i odwołania do przestrzeni adresowej zbioru modułów tworzących program. Odwołania te mają formę adresów logicznych, a ich allokacja wykonywana jest w trakcie procesu łączenia, który nie może zmieniać kodów operacji. Jest to podstawowa funkcja programu łączącego, poza uzupełnieniem grupy modułów użytkownika o moduły biblioteczne w celu utworzenia kompletnego programu. Możliwa jest zatem ustalenie jednolitej formy reprezentacji wyrażeń adresowych, niezależnej od typu mikroprocesora, a zatem i języka pośredniego. W języku tym zapisane są moduły wynikowe asemblerów i moduły biblioteczne. Specyficzne własności określonego typu mikroprocesora (kod rozkazu i format rozkazu, rozmiar pola adresowego) traktuje się jak stałe, nie podlegające konwersji.

Język wysokiego poziomu jest z definicji niezależny od sprzętu. Przez kompilację należy dostosować tekst modułu źródłowego do właściwości sprzętu, na którym ma być wykonany (wyprowadzenie modułu - obniżenie poziomu abstrakcji). By ułatwić realizację taj funkcji w systemie wspomagania

programowania zorientowanym na różne konfiguracje sprzętowe (różne mikroprocesory) przyjmiemy następującą koncepcję. Zadaniem kompilatora języka wysokiego poziomu będzie wyprowadzenie modułu wykonywalnego w pewnym abstrakcyjnym procesorze danym listą rozkazów. Implementację tego procesora w postaci fizycznej osiągnąć można przez:

- konstrukcję procesora dla danej listy rozkazów,
- implementację danej listy rozkazów w postaci mikroprogramów dla dostępnego procesora mikroprogramowalnego,
- implementację funkcji opisanych daną listą rozkazów w formie zbioru procedur dla konkretnego procesora i interpretację treści modułu pod kontrolą odpowiedniego programu sterującego (interpretera),
- kompilację treści modułu na język fizycznie istniejącego procesora.

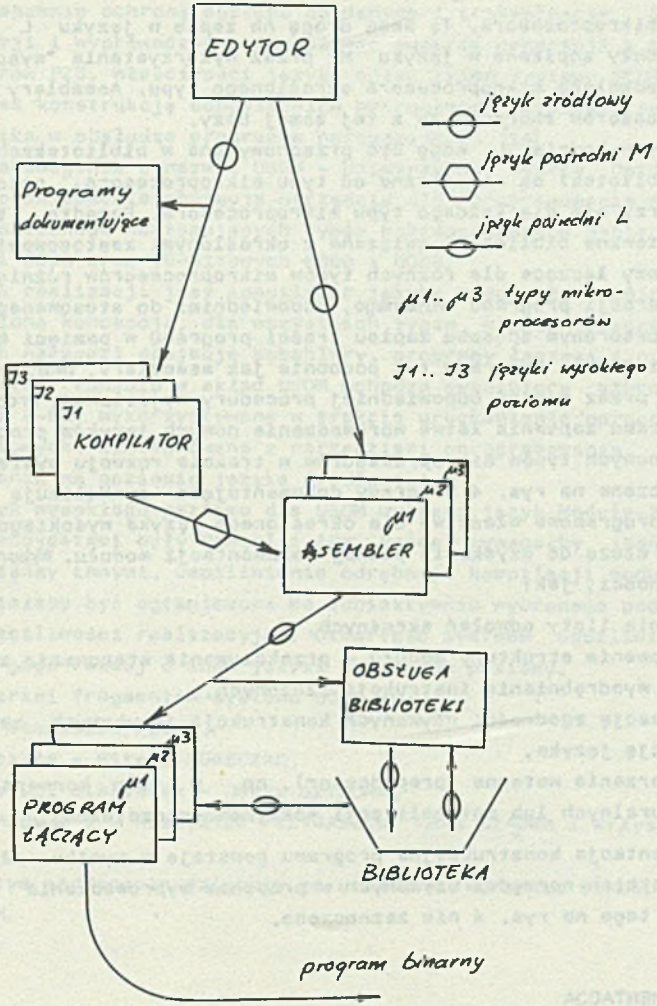
Pierwsze dwa warianty pominiemy, gdyż dotyczą konstrukcji sprzętu, a to nie wiąże się ze wspomaganiem programowania. Technika interpretacji jest powszechnie stosowana w zastosowaniach mikroprocesorów (P - kod dla języka Pascal, interpretery języka BASIC). Wprowadza ona jednak znaczne narzuty czasowe. Ponadto, używanie różnych języków programowania jest w tym przypadku sztuczne i utrudnione. Natomiast obniżenie poziomu abstrakcji modułu przez dodatkową kompilację, przy istnieniu sprawnego makroasemblera, sprowadza się do opracowania zestawu makroinstrukcji odwzorowujących operacje elementarne procesora abstrakcyjnego w operacje procesora fizycznie istniejącego. Należy przy tym zauważyć, że pewne operacje procesora abstrakcyjnego w dalszym ciągu będą musiały być interpretowane przez procedury wywoływane w trakcie wykonywania modułu. Dotyczy to w szczególności operacji na zmiennych o typie nieelementarnym w sensie listy rozkazów procesora fizycznego - operacja zmiennoprzecinkowa i tzw. funkcje standardowe. Ponadto operacje wejścia/wyjścia i ogólniej wywołania systemu operacyjnego również zawsze są interpretowane. Zatem dla implementacji tego wariantu, oprócz zestawu makroinstrukcji, opracować należy grupę modułów (procedur, funkcji) standardowych języka wysokiego poziomu dla określonego mikroprocesora i systemu operacyjnego.

Oczywiste jest, że zdefiniowanie funkcji procesora abstrakcyjnego jest równoważne opracowaniu pewnego języka o pośrednim poziomie abstrakcji.

Przyjęcie tej koncepcji pozwala ponadto na w miarę proste implementowanie różnych języków programowania. Kompilatory tych języków mają wspólny język wynikowy i w dużej części wspólne biblioteki modułów standardowych. Przykładem takiego rozwiązania może być system firmy Whitesmiths [13].

5. STRUKTURA SYSTEMU

Przedstawione wcześniej rozważania stanowią podstawę do zaproponowania struktury uniwersalnego systemu wspomagania programowania mikroprocesorów, przedstawionej na rys. 4.



Rys. 4. Struktura uniwersalnego systemu wspomagania produkcji oprogramowania mikroprocesorów

Edytor służy do zapisu treści modułów na nośniku maszynowym i wprowadzania zmian w tej treści. Jest to edytor uniwersalny. Moduły zapisane w językach wysokiego poziomu są poddawane kompilacji, przy czym wszystkie kompilatory wykorzystuję wspólny język pośredni M dla zapisu wyniku kompilacji [6]. Język ten jest niezależny od typu mikroprocesora docelowego. Moduły zapisane w języku symbolicznym pewnego mikroprocesora przekształcane są przez asembler i zapisane w języku pośrednim L [5]. Dane wykorzystywane w procesie łączenia modułów zapisywane są w formie niezależnej

od typu mikroprocesora. Tą samą drogą na zapis w języku L przekształcane są moduły zapisane w języku M przez wykorzystanie "wyspecjalizowanej" wersji asemblera mikroprocesora określonego typu. Asemblery dla różnych mikroprocesorów tworzone są z tej samej bazy.

Moduły w kodzie L mogą być przechowywane w bibliotekach. Programy obsługi biblioteki są niezależne od typu mikroprocesora, natomiast biblioteki tworzy się dla każdego typu mikroprocesora. Ponadto tworzyć można specjalizowane biblioteki związane z określonymi zastosowaniami.

Programy łączące dla różnych typów mikroprocesorów różnią się procedurami generacji programu binarnego, odpowiednimi do stosowanego w systemie mikroprocesorowym sposobu zapisu treści programu w pamięci komputera (np. w pamięci PROM). Programy te, podobnie jak asemblery, tworzone są ze wspólnej bazy przez montaż odpowiedniej procedury wyjściowej. Proponowana struktura systemu zapewnia łatwe wprowadzanie nowych języków programowania jak również nowych typów mikroprocesorów w trakcie rozwoju systemu.

Zaznaczone na rys. 4 programy dokumentujące symbolizują specjalne narzędzia programowe właściwe dla określonego języka wysokiego poziomu. Programy te służą do uzyskania pełnej dokumentacji modułu. Wykonywać mogą takie czynności, jak:

- tworzenie listy odwołań skrośnych,
- odwzorowanie struktury modułu - przekazywanie sterowania, zanurzanie procedur, wyodrębnianie instrukcji złożonych,
- weryfikację zgodności używanych konstrukcji językowych ze standerdową definicją języka,
- przetwarzanie wstępne (preprocesor), np. w celu konwersji instrukcji strukturalnych lub optymalizacji maszynowo-niezależnej.

Dokumentacja konstrukcyjna programu powstaje w wyniku złożenia raportów wszystkich narzędzi używanych w procesie wyprowadzania programu. Mechanizmu tego na rys. 4 nie zaznaczono.

6. IMPLEMENTACJA

System wspomagania produkcji oprogramowania mikroprocesorów o strukturze jak na rys. 4 został wdrożony w Laboratorium ODRA 1305 Instytutu Informatyki Czasu Rzeczywistego Politechniki Śląskiej w ramach prac badawczych, prowadzonych w latach 1981-1983 na zlecenie problemu węzłowego 06.4.

System jest zaimplementowany dla komputera ODRA-1305, pracującego pod kontrolą systemu operacyjnego GEORGE-3. Jako język implementacji wybrano Pascal dla zapewnienia ewentualnej przenośności systemu na inne komputery. Środowisko systemu operacyjnego GEORGE-3 stwarza istotne ułatwienia we wdrażaniu i eksploatacji systemu wspomagania produkcji oprogramowania mikroprocesorów. Moduły źródłowe i moduły z nich wyprowadzone przechowywane są w zbiorach PZS. Do edycji używa się edytora systemu. System pli-

ków (PZS) zapewnia ochronę dostępu do danych i archiwizację. Dla rejestracji rewizji i wyprowadzeń można używać numerów generacji i kodów językowych zbiorów PZS. Właściwości języka opisu zadań systemu GEORGE-3 umożliwiają przez konstrukcję odpowiednich makrozleceń zminimalizowanie udziału użytkownika w obsłudze programów narzędziowych [14].

System wspomagania o nazwie USOM - Uniwersalny System Oprogramowania Mikroprocesorów obecnie obejmuje narzędzia dla programowania na poziomie języka symbolicznego następujących typów mikroprocesorów 8-bitowych: 8080, 8085, Z-80 i 6800 oraz 16-bitowych 8086 i 8088.

W trakcie realizacji jest kompilator języka Moduła-2, wspólny, zgodnie z przedstawioną koncepcją, dla wszystkich typów mikroprocesorów. Zespół opracowanych narzędzi obejmuje asemblery, programy łączące i programy obsługi bibliotek. Ponadto w skład USOM wchodzą symulatory mikroprocesorów 8080, 8085 i Z-80, wykorzystywane w trakcie uruchamiania opracowanych programów użytkowych, zintegrowane z narzędziami oprogramowania. Pozwala to na uruchamianie na poziomie języka źródłowego.

Jako język wysokiego poziomu dla USOM wybrano język Moduła-2 [15]. Pozostałymi kandydatami były Pascal i Ada. Pascal wymagałby jednak uzupełnień dla, między innymi, umożliwienia odrębnej kompilacji modułów. Z kolei Ada musiałaby być ograniczona do subiektywnie wybranego podzbioru, za względu na możliwości realizacyjne. Otwartość systemu umożliwi jednak w przyszłości jego rozwój o inne języki wysokiego poziomu.

Realizatorami fragmentów systemu USOM są:
asemblery - Stanisław PUCKA,
programy łączące - Witold JURECZKO,
programy obsługi bibliotek - Jerzy SZYMURA,
kompilator Moduła-2 - Władysław PIETRASZEK, Adam CIĄGWA i Krzysztof GROYECKI.

Kolegom tym serdecznie dziękuję za udział w tworzeniu koncepcji i realizacji USOM.

LITERATURA

[1] CRZ80 - BSO Relocating Cross Assembler
MREF - BSO Cross Reference Program
MLINK - BSO Cross Linkage Editor
MLIB - BSO Librarian
OBJCNV - BSO Object File Conversion Utility
SIZ80 - BSO Simulator/Debugger
Boston System Office, 1982.

[2] Cosserat D.: Micro Sim - a new approach to program development, Microprocessors and Microsystems vol 2, no 2 1979.

[3] Jenseth J., Hebert J.: Intermetrics Cross - Compiler remedies "Brute-Force" approach to microprocessor software development, Intermetrics Inc, Technical Backgrounder, 1980.

[4] Nałęcki K., Jureczko W., Pucka St.: Opracowanie skrośnego przenośnego oprogramowania rozwojowego systemów mikroprocesorowych. Opracowanie założeń konstrukcyjnych systemu oprogramowania i techniki łączenia modułów wynikowych. Raport z realizacji zadania 01/04/1.1 pr. węzłowy 06.4, Instytut Informatyki Czasu Rzeczywistego Politechniki Śląskiej, Gliwice 1981.

[5] Nałęcki K., Jureczko W., Pucka St., Szymura J.: Opracowanie skrośnego przenośnego oprogramowania rozwojowego systemów mikroprocesorowych z przykładową implementacją na maszynie cyfrowej ODRA-1305. Program łączący moduły wynikowe. Programy obsługi biblioteki modułów wynikowych. Raport z realizacji zadania 01/04/1.2 pr. węzłowy 06.4. Instytut Informatyki Czasu Rzeczywistego Politechniki Śląskiej, Gliwice 1982.

[6] Nałęcki K., Ciągwa A., Groyecki K., Jureczko W., Pietraszek W., Pucka St.: Rozwój systemu oprogramowania mikroprocesorów USOM. Raport z realizacji zadania 01/04/1.3 pr. węzłowy 06.4. Instytut Informatyki Czasu Rzeczywistego Politechniki Śląskiej, Gliwice 1983.

[7] Willis R.R., Aides: Computer aided design of software systems w Software Engineering Environments, ed. H. HUENKE North-Holland 1981.

[8] Handlykken P., Nygaard K.: The DELTA system description language: motivation, main concept and experience from use, w Software Engineering Environments, ed. H. HUENKE North - Holland 1981.

[9] Spier M.J., Gutz S., Warsserman A.I.: The Ergonomics of Software Enginnering - description of the problem space, w Software Engineering Environments, ed. H. HUENKE North - Holland 1981.

[10] Sperry Univac 1100 Series Conversational Time Sharing System 7940 Rev. 4. Program Control - Language Prescanner COBOL, FORTRAN, Sperry Univac 1978.

[11] Cheatham T.E.: Comparing Programming Suport Environments ed H. HUENKE North - Holland 1981.

[12] Pucka St.: Automatyczna generacja kompilatorów języków asemblera, Zeszyty Naukowe Politechniki Śląskiej, seria Informatyka, nr 7, Gliwice 1984.

[13] WHITESMITHS SOFTWARE CATALOG, Whitesmiths Ltd, 1981.

[14] Borak S., Klaczak J., Korczak S., Płoski Z.: System operacyjny George-3, WNT, Warszawa 1981.

[15] Wirth N.: Programming in Modula-2, Springer-Verlag, 1983.

Recenzent: Doc. dr hab. inż. Adam Wolisz

Wpłynęło do redakcji: 13.03.1984 r.

ПОДДЕРЖКА ПРОИЗВОДСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРОПРОЦЕССОРОВ - ИДЕЯ УСОМ

Р е з ю м е

В статье представлена общая характеристика проблем производства прогаммного обеспечения, в частности для микропроцессоров. Оговариваются также свойства применяемого инструментального программного обеспечения. Обсуждается использование разных языков программирования для различных типов микропроцессоров. Охарактеризована область внедрения предложенной в виде УСОМ - универсальном системы программирования для микропроцессоров.

MICROPROCESSOR SOFTWARE DEVELOPMENT AID
AN IDEA OF USOM

S u m m a r y

In this paper general characteristics of a software development process, in particulary for the microprocessors, and features of applicable software tools have been stated. Different programming languages application for different microprocessors types has been discussed. Structure of software development aid system for microprocessors has been proposed. At the end, implementation of suggested ideas in the form of USOM - universal system for microprocessors software making has been presented.