ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1984

Seria: INFORMATYKA z. 7 Nr kol. 809

Stanisław PUCKA

Instytut Informatyki Czasu Rzeczywistego
Politechniki Śląskiej

AUTOMATYCZNA GENERACJA KOMPILATORÓW JĘZYKA ASEMBLERA

Streszczenie. W artykule omówiono system usprawniający proces tworzenie kompilatorów języka asemblera dla różnych komputerów. System ten został opracowany i jest używany w laboratorium ODRA-1305 Instytutu Informatyki Czasu Rzeczywistego Politechniki Śląskiej.

WSTĘP

Projekt USOM [5] (Uniwersalny System Oprogramowania Mikroprocesorów) realizowany w Instytucie Informatyki Czasu Rzeczywistego jest to zestaw narzędzi programowych ułatwiających oprogramowanie systemów mikroprocesorowych.

Zasadniczą cechę tego systemu jest wykorzystanie dużego komputera zwanego dalej komputerem źródłowym do tworzenia oprogramowania systemów mikroprocesorowych (komputerów docelowych).

Realizacja projektu oparta była na dwóch głównych założeniach:

- uniwersalności, czyli niezależności od komputera docelowego,
- przenośności, czyli niezależności od komputera źródłowego.

W skład systemu wchodzą następujące elementy:

- kompilator języka wyższego rzędu,
- kompilatory języka asemblera,
- programy łączące,
- podsystem obsługi bibliotek.

Produktem końcowym otrzymanym w wyniku użycia systemu USOM jest binarny program akceptowany przez oprogramowanie podstawowe komputera docelowego lub urządzenie programujące pamięci typu ROM.

Założenie przenośności zostało zrealizowana w ten sposób, że wszystkie elementy systemu zostały napisane w języku programowania Pascal, dzięki czemu cały system jest przenośny na tyle, na ile przenośne są programy napisane w tym języku.

Założenie uniwersalności wymaga dokładniejszego omówienia, w szczególności w odniesieniu do kompilatorów języka asemblera. Język ten jest z definicji silnie uzależniony od komputera, dla którego został stworzony.

Jedynym więc zadowalającym sposobem realizacji postulatu uniwersalności systemu USOM na poziomie języka asemblera jest dostarczenie oddzielnego kompilatora dla każdego obsługiwanego komputera. Przy tak postawionym zadaniu nasuwa się myśl, by zautomatyzować proces tworzenia kompilatorów języków asemblera dla różnych komputerów.

Mechanizm taki został opracowany i wykorzystany przy realizacji projektu USOM. W dalszej części omówiony jest on bardziej szczegółowo. Omówienie to poprzedzone jest próbą usystematyzowania pojęć z dziedziny programowania na poziomie języka asemblera.

1. JĘZYK ASEMBLERA

Przedstawione w tym punkcie spostrzeżenia zostały dokonane między innymi na podstawie analizy firmowych języków asemblera dla następujących mikroprocesorów ([1], [2], [6], [7]):

INTEL 8085,
INTEL 8086,
ZILOG Z80,
TEXAS INSTRUMENTS TMS 9900,
MOTOROLA M6800,
oraz języka programowania PLAN-4 komputerów serii ICL 1900.

Analiza ta wykazała, że istnieją pewne ogólne charakterystyczne cechy tego typu języków.

Są one następujące:

- Język asemblera odwzorowuje architekturę komputera oraz niektóre właściwości systemu operacyjnego (jeśli jest używany), który w tym przypadku może być traktowany podobnie jak sprzęt.
- Język taki powinien z jednej strony zapewnić możliwość pełnej kontroli programisty nad generowanym kodem binarnym, a równocześnie dostarczyć mechanizmów automatyzujących możliwie wiele czynności nieistotnych z punktu widzenia efektu końcowego.
- Istotną cechą jest możliwość używania w treści programu źródłowego nazw symbolicznych reprezentujących kody operacji, adresy, stała okresu kompilacji i inne obiekty.

W każdym języku programowania można wyróżnić trzy jego aspekty ([3]):

- składnię,
- semantykę,
- pragmatykę.

Przeprowadzona pod tym kątem analiza różnych języków pozwoliła wysnuć szereg wniosków istotnych z punktu widzenia konstrukcji generatora kompilatorów języka asemblera.

Składnia

Jest to zbiór reguł tworzenia poprawnych konstrukcji (programów) w danym języku. W przypadku języków asemblera łatwo zauważyć, że składnia większości z nich jest do pewnego poziomu prawie identyczna.

Podstawowym pojęciem zdefiniowanym w języku jest instrukcja. Instrukcja zajmuje zwykle jedną linię programu źródłowego i ma następującą składnię:

[ETYKIETA] [[KOD OPERACJI] [OPERANDY]] [KOMENTARZ]

W nawiasach [ ] zawarte są opcjonalne elementy instrukcji.

W starszych językach instrukcja ma na ogół format pozycyjny, tzn. linia podzielona jest na pola określonej pozycji przeznaczone na poszczególne elementy instrukcji. W nowych językach w większości przypadków stosowany jest format swobodny. Reguły składni na poziomie elementów instrukcji są wtedy zbliżone lub identyczne z niżej podanymi uogólnionymi zasadami.

ETYKIETA, jeśli występuje, rozpoczyna się w pierwszej kolumnie linii i jest zakończona odstępem lub znakiem dwukropka. W tym drugim przypadku nie musi rozpoczynać się w pierwszej kolumnie.

KOD OPERACJI musi być poprzedzony przynajmniej jednym znakiem odstępu, a jeśli występuje etykieta zakończona dwukropkiem, odstęp nie jest wymagany.

OPERANDY, jeśli występują, oddzielone są od kodu operacji przynajmniej jednym znakiem odstępu, a pomiędzy sobą przecinkami.

KOMENTARZ rozpoczyna się od wyróżnionego znaku komentarza.

Łatwo zauważyć, że tak zdefiniowane reguły składni mogą służyć zarówno do analizy instrukcji w zapisie swobodnym, jak i pozycyjnym.

Poza podaną definicją składni instrukcji, w większości języków można wyróżnić jeszcze kilka elementów o podobnych regułach składni.

SYMBOL - jest to ciąg znaków rozpoczynający się od litery i zawierający znaki liter i cyfr.

LICZBA - jest to ciąg znaków rozpoczynający się cyfrą 0 .. 9, zawierający znaki cyfr (lub liter A .. F dla liczb szesnastkowych) i zakończony literą określającą typ liczby.

OPERATOR - jest to jeden z wyróżnionych znaków (np. + -) lub symboli (np. AND OR NOT).

ŁAŃCUCH - jest to ciąg znaków ograniczony apostrofami.

Jedynymi elementami instrukcji, których składni nie można uogólnić dla różnych języków są operandy. W zależności od języka mają one różną postać, ściśle związaną ze specyfiką architektury komputera. Przykładem może być zestawienie niektórych dopuszczalnych konstrukcji w różnych językach:

Z80	(IX + 5)	IX jest nazwą rejestru
M6800	#SYMBOL	adresacja zerowej strony pamięci
TI 9900	* R 3 +	adresacja pośrednia z autoinkrementacją
I 8086	SYMBOL [SI] [BP]	podwójna indeksacja.

Semantyka

Semantyka opisuje znaczenie poszczególnych konstrukcji języka.

Analiza różnych języków z tego punktu widzenia wykazała, podobnie jak w przypadku składni, istnienie wielu podobieństw w tych językach.

Najważniejsze z nich są podane niżej.

Etykieta reprezentuje adres pamięci, gdzie został umieszczony kod binarny odpowiadającej jej instrukcji.

Kod operacji jest nazwą mnemoniczną instrukcji z listy rozkazów komutera.

Operandy uściślają funkcję instrukcji określonej przez kod operacji.

Definicja języka w zakresie semantyki sprowadza się w zasadzie do określenia zależności między kodem operacji i postacią operandów instrukcji a odpowiadającym jej kodem binarnym i jest ściśle uzależniona od komputera, dla którego język jest definiowany.

Pragmatyka

Pod tym pojęciem rozumie się wszystkie inne aspekty języka, a w szczególności jego walory użytkowe. Pragmatyczne cechy języka są ściśle związane zarówno ze składnią i semantyką, jak i kompilatorem tego języka.

Poniżej omówione są najistotniejsze z punktu widzenia techniki programowania cechy języka i związanego z nim kompilatora.

Modularność

Jest to właściwość umożliwiająca zestawienie kompletnego programu z fragmentów zwanych modułami.

Moduł może być zdefiniowany jako najmniejsza niezależnie kompilowana część programu. W przypadku języka asemblera moduł jest pojęciem zbliżonym do fortranowskiego segmentu.

Modularność umożliwia niezależne tworzenie i testowanie fragmentów programu będących funkcjonalną całością, tworzenie bibliotek programów i znaczne usprawnienie pracy zespołu programistów.

Gospodarka pamięcią

Jest to jedna z podstawowych funkcji kompilatora. Polega ona na wyznaczeniu adresów poszczególnych instrukcji programu i wiązaniu tych adresów z nazwami symbolicznymi. Często funkcje te są realizowane dwustopniowo: przez kompilator i współpracujący z nim program łączący. Jest to również rozwiązanie przyjęte w systemie USOM.

Zagadnienia gospodarki pamięcią związane są z często występującym podziałem pamięci komputera na różne klasy. Podział ten wynika zarówno z architektury komputera, techniki programowania, jak i technicznej realizacji pamięci.

Przykładowo można wymienić:

- pamięć dla stosu programu,
- wspólne obszary danych,
- pamięć typu RAM,
- pamięć typu ROM.

W systemie USOM zaproponowano ujednolicenie pojęć związanych z gospodarkę pamięcią i uniezależnienie jej od komputera docelowego.

Wprowadzone zostały pojęcia sekcji pamięci i obszaru pamięci.

Pojęcie sekcji występuje na poziomie programu źródłowego. Sekcja identyfikowana jest przez nazwę symboliczną określoną przez programistę. Pojęcie obszaru pamięci występuje wyłącznie na poziomie łączenia modułów i generacji kodu binarnego. Obszar pamięci określony jest przez absolutne adresy początku i końca obszaru. W ramach modułu można definiować treść pamięci dla dowolnej ilości sekcji. W fazie łączenia modułów następuje grupowanie treści sekcji o tych samych nazwach pochodzących z różnych modułów i alokacje kodu wynikowego w określonym obszarze pamięci.

Należy zwrócić uwagę, że to rozwiązanie jednocześnie zapewnia relokowalność (przesuwalność) programu.

Makroinstrukcje

System makroinstrukcji umożliwia automatyczną generację pewnych fragmentów programu na podstawie wzorca zdefiniowanego przez programistę. W systemie USOM został zdefiniowany prosty a równocześnie, jak się wydaje, efektywny system makroinstrukcji.

Dyrektywy

Są to instrukcje nie związane z listę rozkazów komputera, a służące do organizacji pracy kompilatora oraz usprawniające proces programowania.

Zestaw dyrektyw jest w większości analizowanych języków podobny. Nietrudno więc określić, ile i jakie dyrektywy powinny być dostępne w kompilatorach systemu USOM.

Pozostałe aspekty pragmatyczne języka związane są z kompilatorem i pozostałymi elementami systemu programowania. Wśród nich należy wyróżnić następujące:

- obszar pamięci zajmowanej przez kompilator,
- szybkość kompilacji,
- diagnostyka błędów,
- dostępne raporty.

W przypadku kompilatorów generowanych automatycznie istotne jest, by narzuty i ograniczenia z tego wynikające nie były zbyt duże.

2. GENERATOR KOMPILATORÓW

Podsystem generacji kompilatorów języka asemblera jest częścią systemu USOM pozwalającą zrealizować założenie jego uniwersalności na poziomie języka ukierunkowanego maszynowo. Efektem wynikowym jest tutaj kompilator wcześniej zdefiniowanego języka.

Zastanówmy się na wstępie, jakie właściwości powinien mieć taki mechanizm.

Niewątpliwie najważniejsza jest możliwość generacji kompilatorów dla dostatecznie dużej klasy języków. Wynika to z faktu, że na ogół dla każdego systemu komputerowego producent zdefiniował jakiś język asemblera i w związku z tym celowe wydaje się, by taki właśnie język został zaimplementowany. W praktyce może to być niekiedy trudne do zrealizowania, dlatego przyjęto założenie, by system generacji zapewniał możliwość uzyskania kompilatora dla dowolnego komputera, co nie oznacza jednak, że dla dowolnego języka. Takie założenie mogło zostać przyjęte, ponieważ dla każdego komputera można zdefiniować dowolnie wiele różnych języków asemblera, a w szczególności taki, który łatwo zaimplementować w procesie automatycznej generacji.

W przeciwnym przypadku, tzn. gdyby żądać możliwości automatycznej generacji dla dowolnego języka, jedynym rozwiązaniem byłoby stwierdzenie, że taki system już istnieje: jest nim jeden z dostępnych w komputerze źródłowym języków programowania, w którym można napisać dowolny kompilator.

Z tym stwierdzeniem wiąże się drugi ważny postulat dotyczący systemu automatycznej generacji. Jest nim wygoda jego użytkowania. Decyduje tutaj ogólnie pojęty nakład pracy niezbędny do otrzymania kompilatora w wersji nadającej się do szerokiego użytkowania.

Charakterystyczną cechą tego typu narzędzi jest, że zmniejszenie nakładu pracy potrzebnego do uzyskania efektu końcowego związane jest na ogół ze zwiększeniem ograniczeń na ten produkt.

W tym konkretnym przypadku należy system generacji tak zaprojektować, by był on wygodny w użyciu, a jednocześnie nie narzucał istotnych ograniczeń na implementowany język.

Poza wyżej wymienionymi pożądanymi cechami systemu automatycznej generacji istotne jest, aby otrzymany kompilator był jak najbardziej efektywny, w szczególności zminimalizowane powinny być narzuty (objętość pamięci i szybkość działania) wynikające ze sposobu jego otrzymania.

Przyjęcie powyższych założeń pozwoliło wybrać koncepcję realizacji systemu po rozważeniu kilku omówionych poniżej możliwości.

1. Użycie odpowiednio skonstruowanego makrogeneratora. W takim rozwiązaniu każda z nazw mnemonicznych języka byłaby zdefiniowana jako makroin-

strukcja. Rozwiązanie to, niewątpliwie interesujące i stosunkowo proste, ma jednak dwie podstawowe wady: narzuca stosunkowo duże ograniczenie na strukturę języka oraz uzyskany w ten sposób kompilator jest mało efektywny (co wynika z zasady działania makrogeneratora).

2. Skonstruowanie kompilatora dla języka o ustalonej składni i strukturze z wymienialnymi tablicami nazw mnemonicznych. Takie rozwiązanie daje efektywny kompilator, a sam system wydaje się być prosty i skuteczny.

Szczegółowe rozważenie tak pomyślanego systemu generacji wykazało, że posiada on jednak zasadnicze wady. Uzyskane kompilatory różnią się jedynie zestawem akceptowanych nazw mnemonicznych, w związku z tym jest to w zasadzie kompilator dla jednego języka w różnych wariantach.

W tym rozwiązaniu implementacja języka dla komputera z bogatą listą rozkazów lub bardziej skomplikowaną architekturą powoduje znaczny wzrost ilości nazw mnemonicznych języka, gdyż jest to jedyny sposób odwzorowania właściwości komputera. Tymczasem w nowych rozwiązaniach można zaobserwować korzystną z punktu widzenia użytkownika języka tendencję do ograniczenia ilości nazw mnemonicznych i odwzorowania szczegółów architektury komputera poprzez wzbogacenie składni operatorów. Typowym przykładem może być tutaj porównanie języków zdefiniowanych dla mikroprocesorów INTEL 8080 i ZILOG Z80.

3. Konstrukcja kompilatora przez zestawienie go z wcześniej przygotowanych i przetestowanych modułów i uzupełnienie go o moduły, które ewentualnie należy stworzyć dla tego konkretnego kompilatora. To ostatnie rozwiązanie zostało przyjęte do realizacji z uwagi na to, że narzuca najmniej ograniczeń na implementowany język, przy stosunkowo niedużej pracochłonności związanej z procesem generacji.

Podstawą systemu generacji jest tzw. szkielet kompilatora. Jest to prawie kompletny kompilator napisany w języku Pascal i przechowywany w postaci źródłowej. Szkielet zawiera następujące procedury:

- inicjacji kompilatora,
- wczytania kolejnej linii programu źródłowego i analizy składniowej do poziomu elementów instrukcji,
- generacji kodu wynikowego,
- organizacji tablic, generacji raportów, sygnalizacji błędów itp.,
- obsługi typowych instrukcji oraz pełnego zestawu dyrektyw,
- obsługi systemu makrogeneracji,
- analizy składniowej pozostałych elementów języka: wyrażeń, liczb, symboli itd.

Uzupełnienie szkieletu o pewne elementy napisane przez użytkownika systemu generacji daje w efekcie źródłowy program kompilatora. Wybór struktury szkieletu wynikł bezpośrednio z wcześniej dokonanych uogólnień dotyczących programowania na poziomie języka asemblera. Szkielet zawiera te elementy, które występują w każdym kompilatorze i są niezależne lub zależą w niewielkim stopniu od akceptowanego języka. Poza tym w szkielecie

umieszczone zostały elementy niezbędne ze względu na wygodę używania systemu generacji, które jednakże determinują pewne aspekty języka.

Najważniejsze z nich są pokrótce omówione.

Analizator składni instrukcji

Jest to procedura uruchamiana po wczytaniu kolejnej linii programu źródłowego, która dokonuje analizy składni instrukcji do poziomu podstawowych elementów, tzn. etykiety, kodu operacji, operandów i komentarza.

Analizator akceptuje instrukcje o składni opisanej w poprzednim rozdziale.

Procedury obsługi dyrektyw i instrukcji

Procedury te są uruchamiane po zakończeniu wstępnej analizy składni instrukcji. Na podstawie mnemoniki (zawartości pola kodu operacji) wybierana jest i następnie wykonywana podporządkowana jej procedura. Treść tej procedury zawiera kontynuację analizy składni operandów instrukcji oraz akcje związane z wystąpieniem tej instrukcji w programie.

W szkielet kompilatora wbudowane są procedury obsługi dyrektyw dostępnych w kompilatorach systemu USOM oraz typowych instrukcji występujących w języku asemblera. Użytkownik generatora musi napisać i dołączyć do szkieletu procedury obsługujące instrukcje o składni i semantyce nie przewidzianej w szkielecie.

System makrogeneracji

System ten jest w całości wbudowany w szkielet. Nie jest przewidziane dokonywanie w nim zmian w fazie generacji. W związku z tym wszystkie generowane kompilatory są wyposażone w jednakowy system makrogeneracji, na ogół różny od wzorcowego.

Obiekty używane w procedurach użytkownika

W treści procedur dołączonych przez użytkownika w fazie generacji są używane pewne obiekty zdefiniowane w szkielecie. Umożliwiają one napisanie tych procedur w sposób prosty i przejrzysty. W czasie generacji możliwa jest w pewnym zakresie zmiana właściwości funkcjonalnych tych obiektów przez definicje odpowiednich stałych, w celu dostosowania ich do wymogów użytkownika.

Typowa procedura użytkownika składa się z kilku do kilkunastu instrukcji w języku Pascal, chociaż dla instrukcji bardziej złożonych, szczególnie tych w których nazwa mnemoniczna obejmuje wiele różnych wariantów kodu binarnego, konieczne jest niekiedy znaczne rozbudowanie odpowiadających im procedur.

Przebieg generacji

Proces generacji kompilatora przebiega wg następującego schematu:

1. Definicja języka.
2. Przygotowanie procedur użytkownika oraz odpowiednich tablic.
3. Edycja tekstu kompilatora.
4. Kompilacja uzyskanego tekstu kompilatora.
5. Inicjacja.
6. Testowania binarnego programu kompilatora.

Poniżej omówione są niektóre bardziej istotne aspekty poszczególnych etapów generacji.

Definicja języka

Kryteria wyboru implementowanego języka wynikają z uwag zamieszczonych w poprzednich rozdziałach.

Należy zwrócić uwagę na fakt, że praktycznie niemożliwa jest pełna kompatybilność języka wzorcowego z implementowanym, co wynika chociażby z różnic sprzętowych. Można jednakże na ogół tak zaprojektować język, by możliwa była przenośność oprogramowania w postaci źródłowej po stosunkowo niewielkich zabiegach edycyjnych.

W niektórych przypadkach system generacji narzuca pewne ograniczenia strukturalne i wtedy może się okazać, że implementacja języka wzorcowego jest niemożliwa lub byłaby zbyt kosztowna, gdyż wiązałaby się z ingerencją w szkielet kompilatora.

Przygotowanie uzupełnień do szkieletu

Aby wygenerować kompilator, użytkownik musi przygotować dwa różne pliki.

1. Zestaw instrukcji w języku Pascal, które zostaną włączone do treści źródłowej kompilatora.

W skład tego zestawu wchodzą:

- instrukcje definiujące stałe kompilatora oraz jeśli taka konieczność, typy i struktury danych używane w procedurach użytkownika;
- procedury obsługi instrukcji (jedna procedura dla jednej lub kilku nazw mnemonicznych).

2. Zestaw danych służących do inicjacji kompilatora.

Są to m.in.:

- tablica definiująca nazwy mnemoniczne i powiązania pomiędzy tymi nazwami a odpowiadającymi im procedurami obsługi;
- tablica symboli zastrzeżonych języka;
- tablica znaków specjalnych.

Edycja

W tej fazie generacji następuje włączenie w tekst szkieletu przygotowanego wcześniej zestawu instrukcji.

Skonstruowany został specjalny system edycji automatyzującej ten proces.

Kompilacja

Uzyskany w poprzedniej fazie tekst źródłowy jest kompilowany standardowym kompilatorem języka Pascal.

Inicjacja kompilatora

Uruchomienie po raz pierwszy po kompilacji binarnego programu kompilatora powoduje uaktywnienie procedury inicjacji. Jej funkcję jest wczytanie przygotowanych wcześniej danych inicjujących i określenie na ich podstawie wartości wewnętrznych struktur danych kompilatora.

Po bezbłędnym zakończeniu fazy inicjacji obraz binarny programu jest gotowym do użytku kompilatorem.

Testowanie kompilatora

Szkielet kompilatora i pozostałe elementy systemu generacji zostały wszechstronnie przetestowane w czasie realizacji projektu. Weryfikacja wygenerowanego kompilatora sprowadza się do kontroli elementów wprowadzonych przez użytkownika generatora. Jak wykazało doświadczenie jest to najbardziej pracochłonny etap generacji, szczególnie w przypadku kompilatorów bardziej złożonych języków.

Dla kompilatorów sporządzonych w czasie realizacji projektu USOM stosowane były dwie różne metody usprawniające proces testowania.

1. W przypadku gdy był dostępny oryginalny kompilator języka wzorcowego, porównywano za pomocą prostego programu binarne wyniki kompilacji tego samego programu testowego.

2. Drugi sposób stosowano, gdy nie był dostępny oryginalny kompilator lub implementowany język na tyle różnił się od wzorcowego, że niemożliwe było stosowanie pierwszej metody.

Polegał on na użyciu niezależnie skonstruowanego programu dekompilatora, tzn. programu, który na podstawie binarnego wyniku kompilacji generuje odpowiadający mu program źródłowy w języku asemblera.

Testowanie przebiegało wg następującego schematu:

- kompilacja programu testowego wygenerowanym kompilatorem;
- konwersja otrzymanego programu binarnego do postaci źródłowej za pomocą dekompilatora;
- ponowna kompilacja otrzymanego programu źródłowego;

- porównanie programów binarnych otrzymanych w czasie obu kompilacji,
- wykrycie różnic pomiędzy tymi dwoma programami binarnymi świadczyło o błędzie w kompilatorze lub ewentualnie w programie dekompilatora.

W przypadku niezależnego skonstruowania dekompilatora i dobraniu odpowiedniego programu testującego istnieje duże prawdopodobieństwo wykrycia tą metodą większości błędów kompilatora.

Konstrukcja programu dekompilatora jest stosunkowo prosta, tak że stosowanie opisanej metody jest opłacalne, a poza tym dekompilator może być używany również do innych celów.

3. OTRZYMANE WYNIKI

W ramach realizacji projektu USOM zostały wygenerowane za pomocą opisanego systemu kompilatory dla następujących mikroprocesorów:

- INTEL 8080/85
- ZILOG Z80
- MOTOROLA M6800
- INTEL 8086.

Jako komputer źródłowy został użyty system ODRA-1300/GEORGE-3. Ogólna charakterystyka tych kompilatorów jest następująca:

- obszar zajmowanej pamięci: 18-23 k słów,
- szybkość kompilacji: ok. 2000 linii/min,
- pełna zgodność z pozostałymi elementami systemu USOM.

Niewątpliwie interesująca jest ocena zmniejszenia efektywności kompilatorów wynikająca ze sposobu ich otrzymania. W obszar kompilatora wchodzą pewne procedury używane tylko raz w fazie inicjacji. Dodatkowo, koncepcja szkieletu kompilatora narzuciła konieczność stosowania w jego treści pewnych uogólnień, co również spowodowało zwiększenie obszaru pamięci zajmowanego przez kompilator i zmniejszenie szybkości działania. Narzuty te są różne dla poszczególnych kompilatorów i można je ocenić na ok. 10-15% (zarówno szybkości kompilacji, jak i obszaru pamięci).

Kompilatory INTEL 8080/85, ZILOG Z80 i MOTOROLA M6800 akceptują języki zbliżone do firmowych.

Najistotniejsze rozbieżności są następujące:

- różne systemy makrogeneracji,
- różnice w szczegółach funkcjonalnych niektórych dyrektyw,
- pewne niezgodności mogą wynikać z różnic strukturalnych pomiędzy systemem USOM a kompilatorem wzorcowym.

Kompilator dla mikroprocesora INTEL 8086 jest jedynym zrealizowanym dla komputera szesnastobitowego.

Implementacja w tym przypadku języka firmowego lub zbliżonego do niego okazała się niemożliwa (i chyba również niecelowa) z kilku powodów. Najważniejszym był fakt, że język firmowy w zasadzie nie jest asemblerem. Struktura jego stawia go w pół drogi pomiędzy językiem asemblera a językiem wyższego rzędu. Jego implementacja wymagałaby zmian zarówno w generatorze, jak i w całym systemie USOM.

W tej sytuacji zdecydowano się na generację kompilatora akceptującego zupełnie inny, specjalnie zdefiniowany w tym celu język. Definicja ta jest zgodna z ogólnymi cechami języków tego poziomu omówionymi w poprzednich rozdziałach. Ze względu na złożoną architekturę mikroprocesora 8086 nakład pracy związany z generacją tego kompilatora był największy.

4. PODSUMOWANIE

Omówiony w artykule system generacji kompilatorów języka asemblera może wydawać się trywialny: jest to kompilator bazowy, z którego po dokonaniu w nim poprawek i uzupełnień otrzymuje się kompilator innego języka. Podejście takie jest zresztą dosyć często stosowane przy produkcji oprogramowania. Jednakże w omawianym systemie generacji poza szkieletem kompilatora istotnym elementem jest zestaw reguł formalizujących ten proces oraz narzędzia, które go automatyzują. Dzięki temu zminimalizowane zostały zakłócenia wynikające z ingerencji w treść szkieletu kompilatora, a sam proces generacji stał się bardziej efektywny.

LITERATURA

[1] CRZ 80 - BSO Relocating Cross Assembler. Boston System Office, 1982.

[2] MCS 86 Macro Assembly Language Reference Manual INTEL Co. 1978.

[3] Nicholls J.E.: Struktura języków programowania. WNT, Warszawa 1980.

[4] Nałęcki K., Jureczko W., Pucka St., Szymura J.: Opracowanie skrośnego, przenośnego oprogramowania rozwojowego systemów mikroprocesorowych. Raport z realizacji zadania 01/04/1.2. IICR, Gliwice 1982.

[5] Nałęcki K.: Wspomaganie produkcji oprogramowania mikroprocesorów - koncepcja USOM. Zeszyty Naukowe Politechniki Śląskiej, Seria Informatyka, Gliwice 1984.

[6] Software Data Book - AMI 6800 microprocessor. AMI Co. 1978.

[7] 990 Computer Family System Handbook. TEXAS INSTRUMENTS Co. 1975.

[8] Wirth N.: Algorytmy + struktury danych = programy. WNT, Warszawa 1980.

Recenzent: Doc. dr hab. inż. Adam Wolisz

Wpłynęło do Redakcji: 12.03.1984 r.

АВТОМАТИЧЕСКОЕ ГЕНЕРИРОВАНИЕ КОМПИЛЯТОРОВ ЯЗЫКА АССЕМБЛЕРА

Р е з ю м е

В статье представлена программная система для эффективного создания компиляторов языка уровня ассемблера разных вычислительных машин. Система внедрена для вычислительной машины ОДРА-1305 в Институте Информатики Реального Времени Силезского Политехнического Института.

AUTOMATIC GENERATION OF ASSEMBLY LANGUAGE COMPILERS

S u m m a r y

In this paper software system for efficient creation of assembly language compiler for different computers has been presented. This system has been implemented on ODRA-1305, computer of IICR.