mgr inż. Jan Sadolewski, Ewelina Szmyd
Politechnika Rzeszowska, Katedra Informatyki i Automatyki

# POLSKI KOMPILATOR ORAZ STEROWNIKI PROGRAMOWANE ZA POMOCĄ JĘZYKA IL NORMY IEC 61131-3

*W pracy przedstawiono prototypowy kompilator języka IL (Instruction List) normy IEC 61131-3. Kompilator jest składnikiem pakietu CPDev opracowywanym w Politechnice Rzeszowskiej. Omówiono jego budowę i metody interpretacji kodu IL do postaci drzew wyrażeń używanych przez kompilator języka ST zawarty w CPDev. Wynikiem kompilacji jest binarny kod pośredni interpretowany przez dedykowane maszyny wirtualne pracujące na sterownikach opartych o różne procesory (AVR, ARM, MCS51) oraz komputerach PC.*

## POLISH COMPILER AND CONTROLLERS PROGRAMMED IN INSTRUCTION LIST LANGUAGE OF IEC 61131-3 STANDARD

*The paper presents a prototype compiler of IL (Instruction List) language from IEC 61131-3 standard. The compiler is a component of CPDev engineering environment for programming of industrial controllers. The CPDev has been developed at Technical University of Rzeszów. IL code conversion into tree expressions to be processed by the CPDev's ST compiler is described. Compilation result is an indirect binary code interpreted by dedicated virtual machines implemented in controllers with different processors (AVR, ARM, MCS-51) and PC computers.*

## 1. WPROWADZENIE

Spośród dostępnej w Polsce aparatury kontrolno-pomiarowej przewagę mają duże i średnie systemy zagranicznych producentów. Polscy wytwórcy wdrażają własne rozwiązania dla małych i niekiedy średniej wielkości obiektów. Systemy sterowania producentów zagranicznych mają własne środowiska inżynierskie programowane według normy IEC 61131-3 [2], która została przyjęta w Polsce w 2004 r. Producenci krajowi stosują własne rozwiązania programistyczne niezgodne z tą normą, co czyni ich urządzenia mało konkurencyjnymi. W celu poprawy tej sytuacji w Politechnice Rzeszowskiej, przy wsparciu MNiSzW powstał pakiet CPDev[1] (*Control Program Developer*). W skład pakietu, którego charakterystykę funkcjonalną omówiono w [9] wchodzą: środowisko do tworzenia projektów, kompilator języka ST, symulator projektów oraz konfigurator zasobów sprzętowych. Do CPDev należy również wieloplatformowa maszyna wirtualna zdolna do wykonywania kodu binarnego, będącego wynikiem kompilacji języka ST do postaci uniwersalnego asemblera VMASM (*Virtual Machine Assembler*). Niniejszy artykuł przedstawia stan prac nad kolejnymi elementami pakietu, tzn. kompilatorem języka IL bazującym na istniejącym kompilatorze ST oraz przekształceniem komputera PC wyposażonego w kartę wejść/wyjść obiektowych w urządzenie typu *softcontroller*.

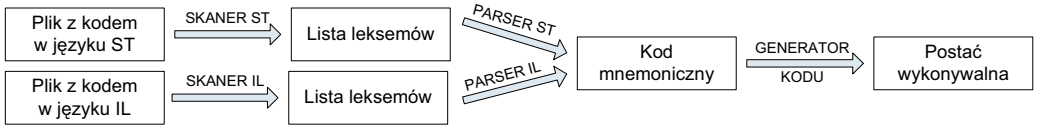
W punkcie 2 omówiono budowę kompilatora języka IL oraz przedstawiono konieczne do współpracy z nim zmiany w kompilatorze ST. Punkt 3 zawiera transformację przykładowego kodu w języku ST do drzew wyrażeń i dalej, do postaci VMASM. Podano równoważny kod w języku IL oraz szczegóły analizy instrukcji języka IL i budowania z nich drzew wyrażeń.

[1] Praca realizowana w ramach projektu rozwojowego MNiSzW nr R02 058 03.

Punkt 4 omawia interpretację specyficznych konstrukcji języka IL z udziałem modyfikatorów. Przedstawiono problemy wynikające z używania języków normy do różnych obiektów POU (*Program Organization Units*). Zaprezentowano autorskie rozwiązania służące bezproblemowej współpracy języków IL i ST w oparciu o dyrektywy i metody analizy kodu. Punkt 6 zawiera opis sterownika SMC produkowanego przez Lumel Zielona Góra, w którym po raz pierwszy wdrożono uniwersalną maszynę wirtualną środowiska CPDev. Scharakteryzowano również sposób wykorzystania jak urządzenia typu *softcontroller* komputera PC z kartą wejść/wyjść krakowskiej firmy InTeCo.

## 2. JĘZYKI PROGRAMOWANIA W NORMIE IEC 61131-3

Norma IEC 61131-3 definiuje pięć języków programowania, w tym dwa tekstowe – ST (*Structured Text*) i IL (*Instruction List*). Ze względu na identyczność sekcji deklaracyjnych obu języków, kompilator IL został zbudowany jako rozszerzenie kompilatora języka ST, którego opis można znaleźć w [5, 6]. Kompleksowy przykład tworzenia systemu sterowania prostą instalacją c.o. w języku ST zaprezentowano w [3]. Do struktury kompilatora ST, wprowadzono alternatywny strumień danych wejściowych w formie pliku z kodem źródłowym w języku IL. Spowodowało to konieczność zmiany budowy wewnętrznej kompilatora do postaci pokazanej na rys. 1.




Rys. 1. Elementy kompilatora języków ST i IL

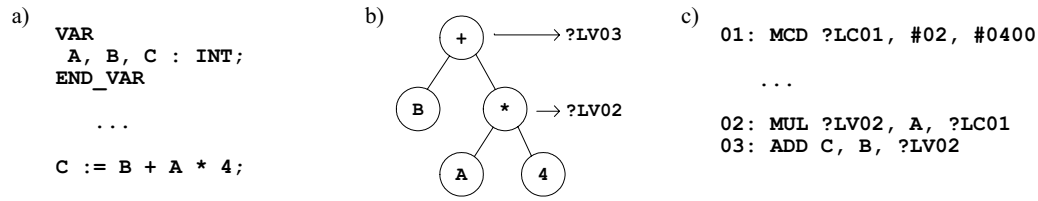
Skaner języka IL został utworzony na bazie skanera ST. Jest on analizatorem leksykalnym, którego zadaniem jest wyodrębnienie z pojedynczych znaków elementarnych jednostek zwanych leksemami. W stosunku do skanera języka ST zmianie uległ zestaw słów kluczowych oraz reakcja na znaki nowej linii, które wskazują koniec instrukcji IL. Wynikiem działania skanera jest lista leksemów, przekazywana do analizatora semantycznego – parsera. Jego zadaniem jest konwersja leksemów do kodu mnemonicznego VMASM, dokładniej omówionego w pracach [7, 8]. Podobnie jak w przypadku skanera, parser IL został zbudowany jako rozszerzenie parsera ST. Pozwoliło to uniknąć redundancji kodu źródłowego obu kompilatorów poprzez wspólne używanie znacznych fragmentów, np. obsługa sekcji deklaracji zmiennych.

Ostatnim elementem kompilatora jest generator kodu, którego zadaniem jest przetworzenie pośredniego kodu zapisanego w asemblerze VMASM do postaci binarnej. Zarówno VMASM jak i generator kodu są wspólne dla obu języków.

## 3. METODY KONWERSJI JĘZYKA IL DO VMASM

W kompilatorze ST zawarte są metody zamiany wyrażeń zapisanych za pomocą funkcji i operatorów do postaci drzew wyrażeń [1]. Zawiera on również procedury konwersji tych drzew do kodu VMASM. Przykład konwersji języka ST do drzewa wyrażenia i kodu VMASM pokazano na rys. 2. W pierwszym etapie zostanie rozpoznana instrukcja podstawienia (rys. 2a) i dla wyrażenia występującego po prawej stronie operatora (:=) zostanie utworzone drzewo (rys. 2b), uwzględniające priorytet operatorów. Wywołuje ono odpowiednie funkcje działające na operandach. Liście drzewa wyrażenia są zmiennymi bądź wartościami stałymi. Funkcje reprezentują wierzchołki nie będące liśćmi. Tak przygotowane drzewo może

ulec optymalizacji polegającej na obliczaniu wartości elementarnych funkcji o stałych argumentach. Obliczalne poddrzewa są zastępowane nowymi wierzchołkami zawierającymi wartości stałe już na etapie kompilacji. Przed konwersją do postaci binarnej stałe występujące w drzewie są zamieniane na zmienne pośrednie i inicjowane. W przykładzie z rys. 2. stałą `4` zamieniono identyfikatorem `?LC01` i zainicjowano w pierwszym wierszu kodu VMASM (rys. 2c). Pojedynczy rozkaz asemblera VMASM składa się z instrukcji elementarnej oraz operandów. Podstawowe instrukcje elementarne tego kodu realizowane przez maszynę wirtualną podano w tabeli 1. Składają się na nie operatory, funkcje arytmetyczne i logiczne, przesunięcia, kilka typów skoków itd. Skompilowany program jest listą takich instrukcji. Operandami są nazwy zmiennych występujących w programie lub wartości stałe. Tymczasowe zmienne generowane przez kompilator posiadają w nazwie znak zapytania '`?`', wykluczający kolizję nazw nadawanych automatycznie z nazwami użytkownika.




Rys. 2. Etapy transformacji kodu języka ST do asemblera VMASM

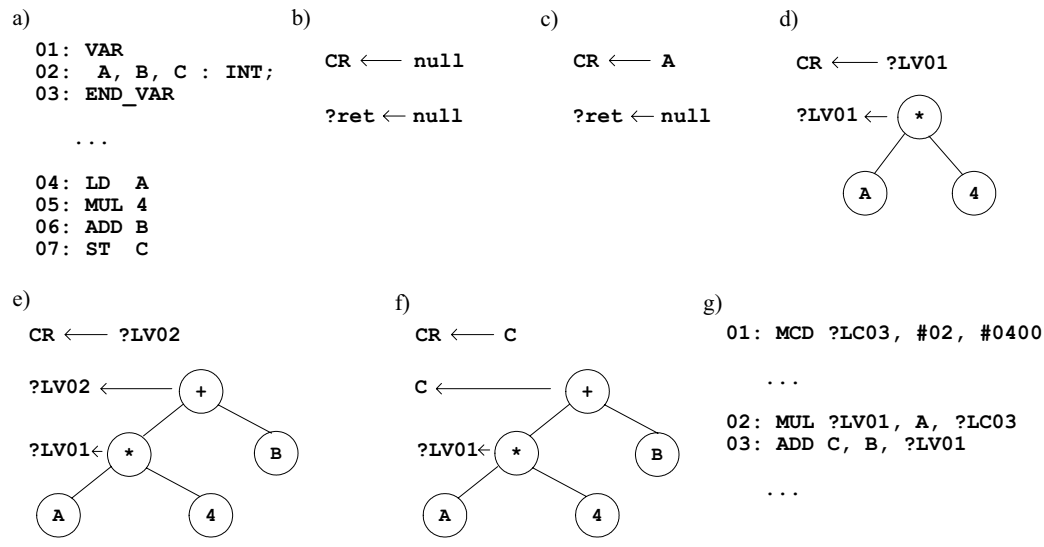
Transformacja drzewa z rys. 2b rozpoczyna się od wierzchołka o największej głębokości nie będącego liściem. Spowoduje to wytworzenie drugiej linii kodu VMASM (rys. 2c) z zapisaniem wyniku do tymczasowej zmiennej lokalnej `?LV02`. Dalsze postępowanie utworzy linię kodu VMASM zapisującą wynik operacji dodawania do zmiennej tymczasowej `?LV03`. Na tym etapie drzewo zostało obliczone a jego wynik znajdujący się w zmiennej `?LV03` zostanie podstawiony do zmiennej `C`. Zmienne tymczasowe w realizacji instrukcji podstawiania zastępowane są zmiennymi docelowymi, co wyjaśnia obecność zmiennej `C` trzeciej linii kodu VMASM w wyniku operacji dodawania. Gdy podstawienie odbywa się pomiędzy zmiennymi użytkownika generowany jest kod kopiowania wartości (instrukcja `MEMCP`).

Tab. 1. Wybrane instrukcje elementarne maszyny wirtualnej

| ***Funkcja*** | ***Opis*** | ***Funkcja*** | ***Opis*** |
|---|---|---|---|
| EXPT | Potęga | AND | Iloczyn logiczny |
| NEG | Negacja arytm. | OR | Suma logiczna |
| SUB | Różnica | XOR | Suma symetryczna |
| MUL | Iloczyn | SHL,SHR | Przesunięcia i obroty bitowe w prawo i lewo |
| DIV | Iloraz | ROL,ROR | |
| ADD | Suma | JMP | Skok bezwarunkowy |
| CONCAT | Konkatenacja | JZ | Skoki warunkowe |
| GT | Większy | JNZ | |
| GE | Większy lub równy | JR | Skok bezwarunkowy względny |
| LE | Mniejszy lub równy | JRN | Skoki warunkowe względne |
| LT | Mniejszy | JRZ | |
| EQ | Równy | RETURN | Powrót z funkcji użytkownika |
| NE | Różny | MCD | Zainicjowanie stałej wartością |
| NOT | Negacja bitowa | MEMCP | Kopiowanie wartości |

W języku IL wykorzystywany jest rejestr (akumulator) oznaczany przez CR [2]. Może on mieścić zmienne dowolnego typu, włączając w to tablice i bloki funkcjonalne. Program napisany w języku IL przypomina asemblerowy kod pewnego bliżej nieokreślonego procesora. Pomimo niskopoziomowego podejścia do programowania, możliwa jest analiza i transformacja kodu IL do postaci drzew wyrażeń interpretowanych przez generator kodu kompilatora ST. Jako uzasadnienie zostanie przedstawiony sposób konstrukcji drzewa wyrażenia na podstawie kodu IL pokazanego na rys. 3a, równoważnego z kodowi ST z rys. 2a.

Po zakończonym etapie analizy leksykalnej lista leksemów jest przekazywana do parsera języka IL. Sposób deklaracji obiektów POU oraz zmiennych jest zgodny z językiem ST. Po deklaracjach następuje sekcja kodu (linie 4-7 rys. 3a). Parser IL tworzy bazowy kontekst parsowania, składający się z:
1) referencji zmiennej będącej obecnym rejestrem CR,
2) przetwarzanej instrukcji,
3) bieżącego drzewa wyrażenia i referencji zmiennej przechowującej jego wynik ?ret.
Wartości początkowe bazowego kontekstu – rejestru CR, drzewa wyrażenia i referencji zmiennej wyniku początkowo ustawione są na wartość null (rys 3b). Wartość null oznacza brak ustalonej referencji zmiennej lub drzewa. Pierwszą napotkaną instrukcją jest LD A, która tutaj nie operacji kopiuje wartości zmiennej A do rejestru CR, a ustawia referencję rejestru CR na zmienną A. Drzewo wyrażenia, oraz referencja zmiennej pamiętającej jego wynik pozostają bez zmian (rys. 3c).




Rys. 3. Etapy transformacji kodu języka IL do VMASM i budowania drzewa wyrażenia

Po napotkaniu wywołania standardowej funkcji mnożenia z argumentem (linia 5 rys. 3a) MUL 4, jest konstruowane drzewo. Korzeniem drzewa staje się operator lub nazwa funkcji, pierwszym argumentem (lewy potomek) zmienna przechowywana w rejestrze CR, a kolejnymi potomkami pozostałe argumenty przekazane w linii podczas wywoływania funkcji. Kompilator tworzy tymczasową zmienną ?LV01 na pamiętanie wartości drzewa. Jej referencja zostaje załadowana do rejestru CR (jako wynik ostatniej operacji). Bieżący kontekst parsowania pokazano na rys. 3d. Kolejną instrukcją jest funkcja ADD z argumentem B, linia 6 rys. 3a. W tym przypadku nowym korzeniem drzewa wyrażenia zostaje ostatnio napotkana funkcja lub jej operator, pierwszym potomkiem korzeń starego drzewa z całym poddrzewem, a drugim po-

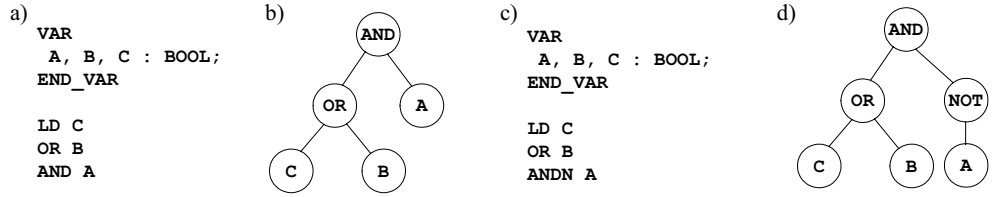
tomkiem argument. W przypadku funkcji wieloargumentowej pozostałe operandy również zostają dopisane jako kolejni potomkowie korzenia drzewa. Wartość drzewa wyrażenia zostaje zapisana do tymczasowej zmiennej ?LV02 oraz trafia jako referencja rejestru CR (rys. 3e). Następną instrukcją jest ST C, której interpretacja zależy od zmiennej wskazywanej przez rejestr CR. Gdy w rejestrze znajduje się zmienna tymczasowa, to wszystkie referencje tej zmiennej tymczasowej zostają zastąpione zmienną będącą argumentem instrukcji ST. Taka sytuacja ma tutaj miejsce – zmienna C zastępuje tymczasową zmienną ?LV02. Gdy w rejestrze CR znajduje się zmienna użytkownika to następuje kopiowanie wartości pomiędzy zmiennymi.

Tak przygotowane drzewo wyrażenia jest równoznaczne z drzewem zapisanym w języku ST. Omówiony poprzednio algorytm konwersji drzewa wyrażenia do kodu VMASM dokona jego transformacji do niemal identycznego kodu, różniącego się tylko nazwami zmiennych tymczasowych. Widać to porównując rys. 2c i 3g.

## 4. INTERPRETACJA MODYFIKATORÓW

W języku IL występują modyfikatory operandów powodujące zmianę ich wartości używanej podczas wykonywania instrukcji. Najczęściej używanymi modyfikatorami są N wyrażający negację argumentu oraz '(' oznaczający początek instrukcji zagnieżdżonych, których wynik będzie później użyty jako operand.

Modyfikator N zastosowany w standardowych funkcjach jak np. AND, powoduje zmianę drzewa wyrażenia poprzez wprowadzenie funkcji NOT aplikowanej do prawego poddrzewa funkcji AND. Na rys. 4b pokazano drzewo wyrażenia dla kodu z rys. 4a, bez modyfikatora N. Rys. 4d przedstawia drzewo wyrażenia dla kodu z modyfikatorem N w funkcji ANDN (rys. 4c).

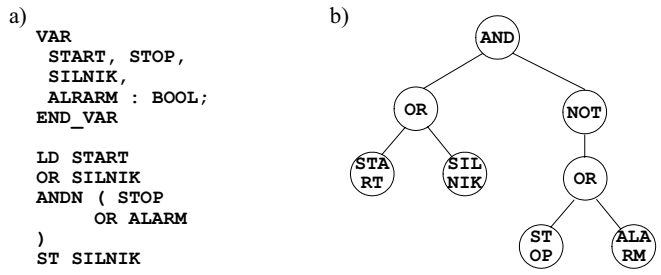



Rys. 4. Drzewa wyrażeń dla kodu bez modyfikatora i z modyfikatorem N

Inna sytuacja następuje w przypadku użycia modyfikatora N dla funkcji STN oraz LDN. W przypadku STN korzeniem drzewa wyrażenia staje się NOT, a jego potomkiem korzeń poprzedniego drzewa wraz z całym poddrzewem. Dalsza operacja przeprowadzana jest podobnie jak w instrukcji ST. Jedyną różnicą jest brak modyfikacji (zamiany) zmiennej tymczasowej przechowywanej w rejestrze CR na zmienną będącą argumentem funkcji STN. Użycie funkcji LDN jest jednoznaczne z wywołaniem kolejno LD, a następnie NOT. Odwrotnie niż w przypadku STN, funkcja LDN nie modyfikuje wartości operandu – tylko tworzy jej lokalną zanegowaną kopię.

Drugi modyfikator '(' oznacza, że bieżąca instrukcja i rejestr CR są zapamiętywane do czasu napotkania ogranicznika ')' tworzącego parę z obecnym. Instrukcje zawarte w nawiasach są wykonywane z „nowym" rejestrem CR, a wartość wyniku używana jako operand w zapamiętanej instrukcji. Kompilator po napotkaniu modyfikatora '(' tworzy nowy kontekst parsowania i inicjuje go następnym operandem, podobnie jak czyni to instrukcja LD. Poprzedni kontekst jest umieszczany na stosie, a kolejne instrukcje są przetwarzane na nowym kontekście do napotkania ogranicznika ')'. Po jego napotkaniu referencja do zmiennej będącej

rejestrem CR zostaje przekazana jako operand dla instrukcji, która znajdzie się na szczycie stosu po zdjęciu z niego obecnego kontekstu. Wiąże się to również z dołączeniem do poprzedniego drzewa wyrażenia, korzenia drzewa z poddrzewami pochodzącego z byłego kontekstu parsowania. Prosty układ Start-Stop, którego kod w języku IL pokazano na rys. 5a, pokazuje efekty zastosowania omówionej metody interpretacji modyfikatora '(' i ogranicznika ')'.




Rys. 5. Układ Start-Stop zapisany w języku IL (a) oraz jego drzewo wyrażenia (b)

## 5. PROBLEMY WYNIKAJĄCE Z ODWOŁAŃ DO KODU ST Z POZIOMU IL

Podczas tworzenia projektów, w których obiekty POU konstruowane są w różnych językach, napotyka się problemy związane z nazwami identyfikatorów, odwołaniami do bezargumentowych funkcji, a także inne.

Pakiet CPDev zapewnia współużywalność bibliotek skompilowanych do kodu VMASM, których źródła mogą być utworzone w różnych językach normy. W trakcie prac nad kompilatorem języka IL napotkano problem kolizji słów kluczowych R i S z nazwami wejść przerzutników RS i SR [2]. Aby umożliwić używanie tych bloków, w kodzie IL podczas rozpoznawania słów kluczowych wprowadzono regułę, że R lub S może być słowem kluczowym, czyli instrukcją IL tylko wtedy, gdy jest pierwszym identyfikatorem w tej linii kodu. Opcjonalnie słowo kluczowe może być poprzedzone etykietami oraz dyrektywami. W pozostałych przypadkach R i S są traktowane jak zwykłe identyfikatory. Planuje się, aby zasadę tę stosować dla wszystkich słów kluczowych języka IL, a nie tylko dla R i S. Pozwoli to na bezpośredni dostęp do nazw zmiennych zdefiniowanych w innych językach.

Kolejnym problemem jest wywołanie w języku IL funkcji bezargumentowej. Jedną z nich jest funkcja maszyny wirtualnej CUR_TIME zwracająca czas systemowy. W przypadku zapisania jej bez operandów zostanie utworzone drzewo, którego korzeniem będzie CUR_TIME, a pierwszym potomkiem zmienna znajdująca się w rejestrze CR. Kompilator będzie wtedy poszukiwał funkcji o sygnaturze odpowiadającej jej nazwie i typowi argumentu wejściowego, która może nie istnieć lub inaczej działać. Aby temu zapobiec została wprowadzona dyrektywa (*$NOARG*), której zadaniem jest oznaczenie wywołania następnej funkcji jako bezargumentowej. Zakres działania dyrektywy ogranicza się tylko do najbliższej funkcji.

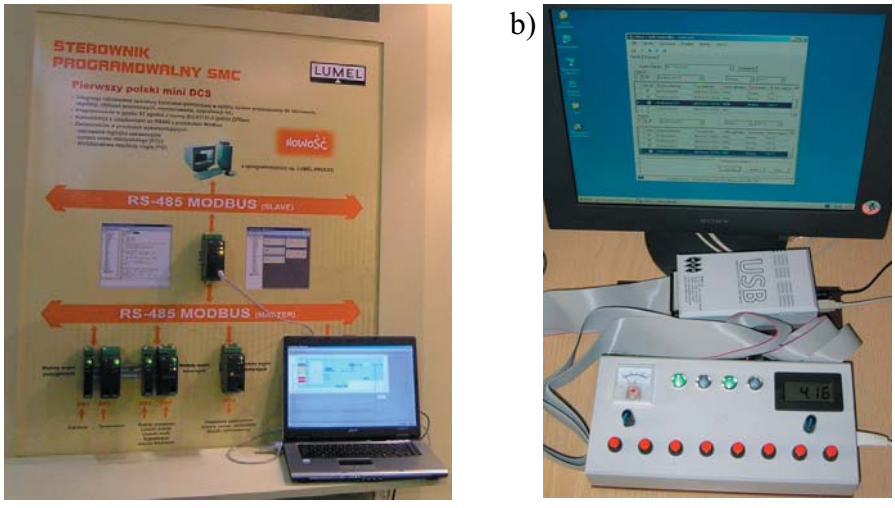
## 6. STEROWNIKI

W asortymencie polskiej aparatury kontrolno-pomiarowej znajdują się sterowniki programowalne i karty pomiarowe, które w połączeniu z innymi urządzeniami mogą tworzyć małe systemy rozproszone. Jednym z takich sterowników jest SMC z LUMELu Zielona Góra (rys. 6), prezentowany na targach Automaticon. Sterownik ten interpretuje kod VMASM zapisany w postaci binarnej, poprzez wbudowaną w oprogramowanie firmowe maszynę wirtualną [8].

Moduł centralny SMC nie posiada własnych wejść ani wyjść obiektowych, dlatego musi być wyposażony w moduły wejścia/wyjścia (np. SM1 do SM5), z którymi komunikuje się za pomocą protokołu Modbus. Stany wejść/wyjść modułów interpretowane są przez program jako zmienne globalne. Szczegółowy sposób konfigurowania komunikacji i programowania sterownika za pomocą języka ST można znaleźć w [3, 4].

Nieco inaczej wygląda sytuacja w przypadku, gdy do sterowania np. procesem laboratoryjnym ma być użyty komputer PC z kartą wejść/wyjść obiektowych. Do tego celu można użyć programu Soft Controller należącego do pakietu CPDev. Do współpracy z kartą potrzebna jest biblioteka (*bridge*) dla platformy .NET łącząca sterownik karty zainstalowany w systemie operacyjnym z aplikacją Soft Controller. Biblioteka ta musi być modułem ładowanym dynamicznie (*assembly*). Etap konfiguracji karty jest podobny jak dla sterownika SMC, ale tutaj następuje przywiązanie zmiennych globalnych do kanałów pomiarowych karty. Pomyślne testy przeszła karta kontrolno-pomiarowa krakowskiej firmy InTeCo, pokazana wraz z panelem testowym na rys. 6b.

a) b)




Rys. 6. a) Stoisko z sterownikiem SMC na targach AUTOMATICON;
b) Testowanie karty wejść/wyjść InTeCo

## 7. PODSUMOWANIE

Przedstawiono prototypowy kompilator języka IL dla pakietu CPDev. Budowa wewnętrzna została oparta na istniejącym kompilatorze języka ST do kodu VMASM. Analiza semantyczna programu w języku IL pozwala na utworzenie drzew wyrażeń, podobnych do powstających przy kompilacji programu ST. Drzewa te są optymalizowane i przekształcane do kodu asemblera VMASM. Instrukcje języka IL mogą zawierać modyfikatory, które przekształcają fragmenty utworzonego drzewa lub zmieniają referencję zmiennej rejestru `CR`. Nieprzemyślane nazewnictwo zmiennych wejściowych/wyjściowych w blokach funkcjonalnych i funkcjach może uniemożliwić ich wykorzystanie w innym języku programowania niż macierzysty. Przedstawiono wybrane metody analizy kodu pozwalające na współużywalność obiektów POU utworzonych w różnych językach. Zaprezentowano dwa wdrożenia maszyny wirtualnej wykonującej skompilowany kod w sterowniku SMC oraz w aplikacji Soft Controller pracującej na komputerze PC z kartą wejść/wyjść obiektowych firmy InTeCo. W przyszłości planuje się pełną integrację kompilatora języka IL z edytorem projektów pakietu CPDev i dalsze rozszerzanie jego funkcjonalności.

## BIBLIOGRAFIA

[1] Cooper K., Torczon L.: Engineering a Compiler. Morgan Kaufmann, San Francisco, 2003.

[2] PN-EN 61131-3 – Sterowniki programowalne. Część 3: Języki programowania. Warszawa, 2004.

[3] Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L.: Programowanie w języku ST sterownika SMC Lumel dla minisystemu rozproszonego. Zgłoszony na konf. Automation 2009.

[4] Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L.: Środowisko programistyczne dla rozproszonych minisystemów kontrolno–pomiarowych. Metody Informatyki Stosowanej, Nr 1, (tom 13) Szczecin 2008.

[5] Rzońca D., Sadolewski J., Trybus B.: Executable form of IEC 61131-3 ST language programs in CPDev environment. Proceedings of the International Multiconference on Computer Science and Information Technology, Wisła 2007.

[6] Rzońca D., Sadolewski J., Trybus B.: Kompilator języka ST normy IEC, 61131-3 na uniwersalny kod wykonywalny. Systemy Czasu Rzeczywistego (red. Z. Huzar, Z. Mazur). Wydawnictwa Komunikacji i Łączności, Warszawa 2007.

[7] Rzońca D., Sadolewski J., Trybus B.: Prototype environment for controller programming in the IEC 61131-3 ST language, ComSIS Vol. 4, No. 2, 2007.

[8] Sadolewski J., Trybus B.: Wieloplatformowa maszyna wirtualna dla systemów sterowania. Modele i Zastosowania Systemów Czasu Rzeczywistego (red. Z. Huzar, Z. Mazur). Wydawnictwa Komunikacji i Łączności, Warszawa 2008.

[9] Stec A., Świder Z., Trybus L.: Charakterystyka funkcjonalna prototypowego systemu do programowania systemów wbudowanych według normy IEC 61131-3. Systemy Czasu Rzeczywistego (red. Z. Huzar, Z. Mazur), Wydawnictwa Komunikacji i Łączności, Warszawa 2007.