Pomiary Automatyka Robotyka, ISSN 1427-9126, R. 26, Nr 1/2022, 23–31, DOI: 10.14313/PAR_243/23

Obsługa typów danych normy PN-EN 61131-3 w architekturze ARM z ograniczeniami dostępu do pamięci

Marcin Hubacz, Jan Sadolewski, Bartosz Trybus
Politechnika Rzeszowska, Wydział Elektrotechniki i Informatyki, ul. Wincentego Pola 2, 35-021 Rzeszów

Streszczenie: W artykule przedstawiono wyniki badań dotyczących obsługi typów danych z normy PN-EN 61131-3 w układach o architekturze ARM. Badania wykonano dla kilku różnych platform sprzętowych z jednostkami centralnymi z serii Cortex-M i Cortex-A. Testy przeprowadzono w oparciu o środowisko CPDev do tworzenia i uruchamiania oprogramowania sterującego. Ze względu na ograniczenia architektury ARM opracowano trzy metody dostępu do pamięci, a wyniki pozwoliły określić najefektywniejszą. W artykule przedstawiono także proponowane rozszerzenie maszyny wirtualnej CPDev z nowymi instrukcjami, dzięki którym operacje na danych w rozwiązaniach o architekturze ARM działają bardziej wydajnie.

Słowa kluczowe: PLC, ARM, PN-EN 61131-3, CPDev

1. Wprowadzenie

Obowiązująca norma PN-EN 61131 (IEC 61131) [1] ustandaryzowała branże automatyki pod wieloma aspektami. Najważniejsze z nich to wymagania i badania dotyczące części sprzętowej, ujednolicenie języków programowania, wymiany informacji oraz sposób projektowania systemów automatyki. Szczególne znaczenie z perspektywy badań ma część trzecia normy dotycząca języków programowania. Oprócz przedstawienia syntaktyki, semantyki i pragmatyki kodowania tekstowego i graficznego oprogramowania, opisuje ona typy danych, architekturę systemów sterowania, model komunikacyjny oraz elementy konfiguracji [1].

Na podstawie licznych doświadczeń przy rozwijaniu oprogramowania CPDev [2, 3] do programowania systemów sterowania w językach standardu PN-EN 61131-3 (ST, FBD, LD, SFC, IL) zauważono duży wpływ rodzaju architektury procesora na czas wykonywania cyklu programu [4]. Wynika to z różnic i ograniczeń procesorów stosowanych do wykonywania kodu maszyny wirtualnej, będącej najważniejszym elementem środowiska wykonawczego (ang. *runtime*). Zróżnicowanie na rynku mikrokontrolerów i mikroprocesorów będących podstawą działania sterowników logicznych utrudnia przygotowanie zoptymalizowanej wieloplatformowej wersji oprogramowania.

Autor korespondujący:
Marcin Hubacz, m.hubacz@prz.edu.pl

Artykuł recenzowany
nadesłany 31.12.2021 r., przyjęty do druku 17.02.2022 r.

Nowoczesne systemy sterowania często wymagają przetwarzania dużych ilości danych różnych typów. W zależności od architektury sprzętowej, na której program jest uruchamiany, wybranie mniej pojemnego typu danych nie musi oznaczać zmniejszenia zajętości obszaru danych lub poprawy wydajności oprogramowania. Przedmiotem badań tego artykułu jest poprawa wydajności pracy systemu sterowania przez zmniejszenie różnic czasu wykonywania instrukcji w zależności od wykorzystywanego typu danych.

Duża część mikrokontrolerów i mikroprocesorów stosowanych w systemach sterowania i wbudowanych oparta jest na architekturze procesorów typu ARM. Ma ona szereg zalet, ale wprowadza również ograniczenia w dostępie do pamięci. W pracy [4] przedstawiono badania wydajności dla układów serii STM32, kładąc nacisk na sprawność wykonywania kodu. W niniejszym tekście przedstawiono wyniki dalszej analizy ograniczeń tej architektury pod kątem przetwarzania danych typów określonych w standardzie PN-EN 61131-3. Badaniami objęto systemy sterowania programowane w środowisku CPDev z maszyną wirtualną działającą w trybie 32-bitowym [4]. Uwzględniono przy tym różne typy rdzeni ARM w układach kilku producentów. Wyniki skłoniły do opracowania rozszerzenia maszyny i kodu pośredniego mającego na celu zminimalizowanie wpływu ograniczeń architektury ARM na szybkość wykonywania programu.

2. Typy danych w systemach sterowania

Przy programowaniu systemów sterowania zgodnie z normą 61131-3 można stosować dane typów prostych (elementarnych) i złożonych. Typy proste można podzielić na kilka grup: liczby całkowite, rzeczywiste, dotyczące daty i czasu, łańcuchy znakowe oraz pozostałe. Część z wymienionych grup może mieć różne wielkości, które skracają lub rozszerzają zakres prze-

chowywanych wartości. Dla przykładu typ INT (liczba całkowita) może mieć warianty takie jak: DINT (liczba całkowita podwójna), LINT (liczba całkowita długa), UINT (liczba całkowita bez znaku), USINT (liczba całkowita krótka bez znaku), ULINT (liczba całkowita długa bez znaku).

W zależności od typu danych kompilator i maszyna wirtualna operują na określonej liczbie bajtów. W tabeli 1 zawarto typy proste zaimplementowane w systemie CPDev wraz z wielkością pamięci potrzebną do przechowywania danych tych typów. Jak widać typowymi rozmiarami zmiennych są 1, 2, 4 i 8 bajtów. Wyjątkiem są łańcuchy znakowe, gdzie rozmiar danych zależy od liczby znaków. Typ ADDRESS jest specjalnym typem w środowisku CPDev używanym wewnętrznie przez procedury i funkcje systemowe. W rozważanej tutaj konfiguracji jest on aliasem dla typu DWORD (adresacja 32-bitowa).

Ponadto dostępne są pochodne typy danych takie jak:
— typ wyliczeniowy,
— typ okrojony.

W normie zdefiniowane są również aliasy dla typów – typy wyliczeniowe i zakresy typów, np. (0.2 ... 12.0). Wartości typów wyliczeniowych i zakresów mogą być reprezentowane za pomocą typów prostych.

Do typów złożonych można zaliczyć tablice i struktury. Na listingu 1 przedstawiono główny fragment programu CPYMEM_WORD w języku ST wykonującego operacje dodawania i reszty z dzielenia (modulo) wartości z dwóch tablic liczb typu WORD. Operacje wykonywane są z użyciem stałej 3.

```
PROGRAM CPYMEM_WORD
 VAR
 buf1 : ARRAY [0..19] OF WORD;
 buf2 : ARRAY [0..19] OF WORD;
 i:INT;
 END_VAR

 FOR i:=0 TO 19 DO
     buf1[i] := (buf2[i] + WORD#3);
     buf2[i] := (buf1[i] mod WORD#3);
 END_FOR

END_PROGRAM
```

Listing 1. Kod w języku ST operujący na tablicach liczb typu WORD
Listing 1. ST language code that operates on arrays of WORD type numbers

Tabela 1. Typy danych w PN-EN 61131 (IEC 61131) i CPDev
Table 1. Data types in PN-EN 61131 (IEC 61131) and CPDev

Nazwa	Typ danych	Rozmiar
Liczby całkowite		
BYTE	bajt	1B (0 ... 255)
WORD	słowo	2B (0 ... 65 535)
DWORD	słowo podwójne	4B (0 ... 2^{32}–1)
LWORD	słowo długie	8B (0 ... 2^{64}–1)
SINT	całkowita krótka	1B (-128 ... 127)
INT	całkowita	2B (-32 768 ... 32 767)
DINT	całkowita podwójna	4B (-2^{31} ... 2^{31}–1)
LINT	całkowita długa	8B (-2^{63} ... 2^{63}–1)
USINT	całkowita krótka bez znaku	1B (0 ... 255)
UINT	całkowita bez znaku	2B (0 ... 65 535)
UDINT	całkowita podwójna bez znaku	4B (0 ... 2^{32}–1)
ULINT	całkowita długa bez znaku	8B (0 ... 2^{64}–1)
Liczby rzeczywiste		
REAL	rzeczywista	4B formatu IEEE-754
LREAL	rzeczywista długa	8B formatu IEEE-754
Data/czas		
TIME	czas trwania	4B (–T#24d20h31m23s ... T#24d20h31m23s)
DATE	data	4B (0001-01-01 ... 9999-12-31)
TIME_OF_DAY	godzina dnia	4B (00:00:00.00... 23:59:59.99)
DATE_AND_TIME	data i godzina	8B (połączenie typów DATE I TIME_OF_DAY)
Inne		
BOOL	logiczny	1B (FALSE/TRUE)
STRING	ciąg znaków ASCII	zmienna długość (każdy znak 1B)
WSTRING	ciąg znaków Unicode	zmienna długość (każdy znak 2B)
ADDRESS	wskaźnik w CPDev	jak DWORD

Listing 2 przedstawia natomiast definicję struktury typu BATTERY_PACK. Zawiera ona siedem pól różnych typów (INT, REAL). Trzy ostatnie pola są zagnieżdżonymi w strukturze tablicami liczb.

```
TYPE BATTERY_PACK
     STRUCT
         NR_BATT   : INT;
         SOC       : REAL;
         SOH       : REAL;
         VOLT_BATT : REAL;
         VOLT_CELL : ARRAY[1..30] OF REAL;
         TEMP_BATT : ARRAY[1..16] OF REAL;
         STATUS    : ARRAY[1..2]  OF INT;
     END_STRUCT
END_TYPE
```

Listing 2. Definicja struktury BATTERY_PACK
Listing 2. Definition of BATTERY_PACK structure

Przechowywanie i wykonywanie dowolnych operacji na zmiennych wymaga od części sprzętowej dostępu do pamięci. Część z typów danych w normie IEC 61131-3 w rzeczywistości jest aliasem dla podstawowych typów danych w językach programowania – na przykład WORD może być reprezentowany jako `uint16_t` w języku C. W zależności od platformy sprzętowej dostęp do pamięci może być wynikiem bardzo zróżnicowanych mechanizmów. Opracowanie jednej uniwersalnej i efektywnej metody dostępu do pamięci dla wielu typów danych na zróżnicowanych platformach sprzętowych jest obiektem badań [5, 6]. Jedną z istotnych cech, którą należy wziąć pod uwagę jest metoda kodowania liczb wielobajtowych, z najmniej znaczącym bajtem jako pierwszym LE (ang. *little endian*) lub ostatnim BE (ang. *big endian*). Używana w niniejszej pracy maszyna wirtualna CPDev może działać w obu trybach, podobnie jak układy ARM. Jednak ze względu na fakt, że domyślnym dla tych ostatnich jest *little endian*, badania przeprowadzone zostały w tym trybie.

3. Działanie systemu sterowania

Podstawowym zadaniem systemu sterowania jest cykliczne wykonywanie oprogramowania opracowanego w jednym z języków normy 61131-3. Wykonywane jest ono ze stałym cyklem odpowiednio dobranym do sterowanego obiektu.

Poprawne działanie programowalnego sterownika logicznego wymaga wykonania kilku dodatkowych punktów. Klasyczny cykl pracy sterownika PLC można zapisać następująco:
— odczyt wejść,
— wykonanie programu,
— ustawienie wyjść,
— zadania komunikacyjne,
— diagnostyka.

Przedstawione wyżej zadania dla sterownika PLC po odpowiedniej konfiguracji generowane są i wykonywane automatycznie, bez udziału programisty. Opracowanie niezawodnego systemu sterowania wymaga oddzielenia algorytmu programu wprowadzanego przez automatyka od zadań realizowanych natywnie, tj. w sposób charakterystyczny dla konkretnej platformy sprzętowej.

Jednostki centralne – mikroprocesory i mikrokontrolery stosowane w systemach sterowania, sterownikach PLC, urządzeniach Internetu Rzeczy czy systemach wbudowanych mają różną architekturę i możliwości. Mogą to być układy typu RISC (ang. *Reduced Instruction Set Computing*), CISC (ang. *Complex Instruction Set Computing*) lub mieszane. Również ich architektura, rozumiana jako wielkość słowa procesora może być odmienna, przy czym powszechne w tych zastosowaniach są układy 8- lub 32-bitowe, choć można spotkać rozwiązania 16- i 64-bitowe.

Różne układy mikroprocesorowe mają swoją charakterystykę, w której kryją się zarówno ich potencjał jak i ograniczenia. Typowy proces tworzenia oprogramowania sterującego dla omawianych zastosowań obejmuje, oprócz przygotowania kodu źródłowego (najczęściej w języku C/C++) przygotowanie kodu wykonywalnego (binarnego) za pomocą dwu narzędzi: kompilatora i programu łączącego (linkera) [6]. Narzędzia te uwzględniają charakterystykę procesora docelowego i generują dla niego optymalny kod, uwzględniający zarówno możliwości jak i ograniczenia układu.

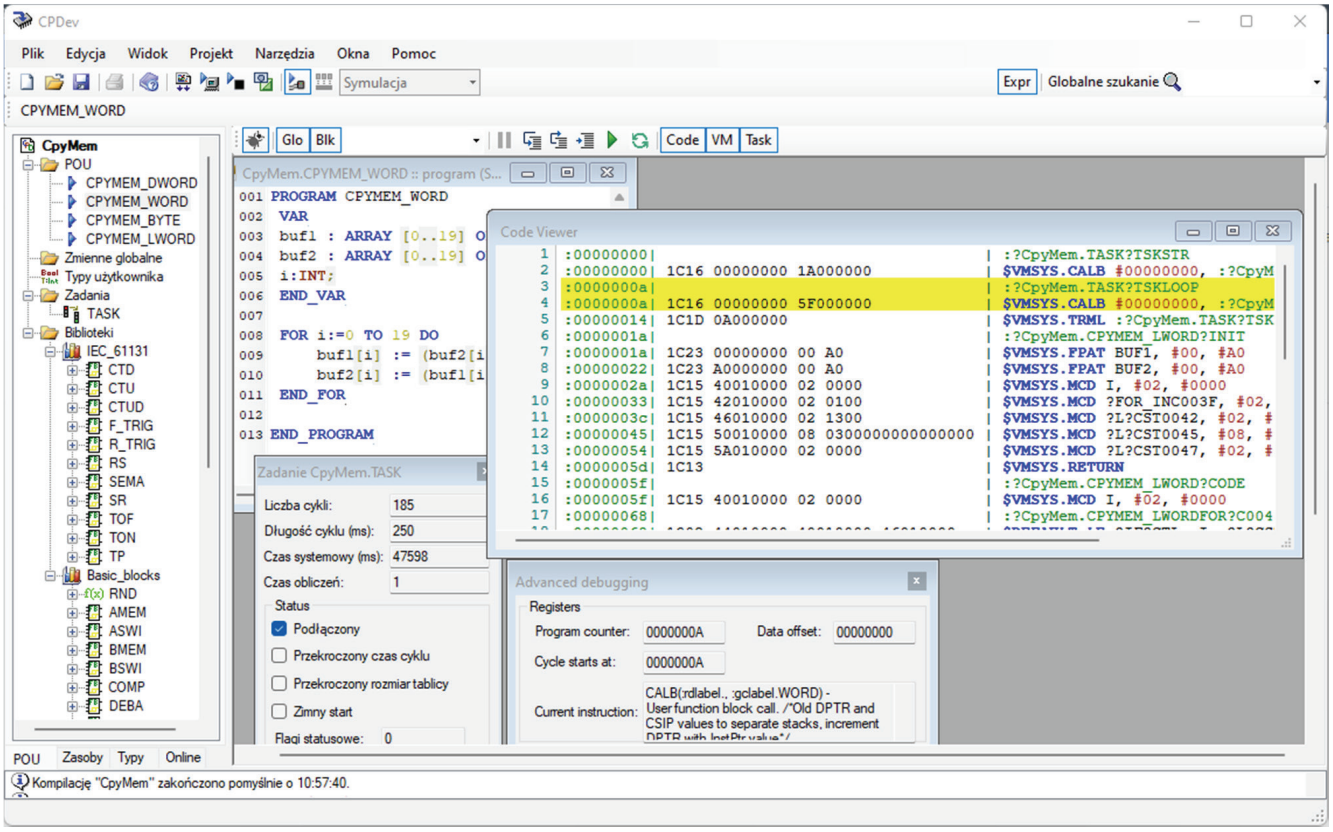
Konieczność zastosowania w procesie przygotowania oprogramowania stosunkowo rozbudowanego zestawu narzędzi (ang. *toolchain*) może być jednak postrzegana jako istotne utrudnienie i spowolnienie procesu wdrażania (ang. *deployment*). Dodatkowo, w przypadku rozproszonych systemów sterowania o niejednorodnej architekturze (sterowniki z odmiennymi procesorami) istnieje konieczność jednoczesnego używania wielu zestawów narzędzi dla różnych procesorów. Pojawia się wówczas problem przenaszalności oprogramowania między różnymi urządzeniami.

Rozwiązaniem tych problemów może być system programowania systemów sterowania oparty o maszyny wirtualne [7, 8]. W tym przypadku generowany kod, zwany pośrednim (ang. *intermediate code*) jest wspólny dla różnych procesorów, dzięki czemu jego wdrożenie jest szybsze, a kod może być łatwo przenoszony na inne platformy [9, 10]. Odbywa się to kosztem wydajności przy wykonywaniu tego kodu, bowiem musi być on przed uruchomieniem zinterpretowany przez środowisko wykonawcze (ang. *runtime*) z maszyną wirtualną.

Co więcej, z racji tego, że kod dla maszyny wirtualnej jest niezależny od platformy, przy jego generowaniu nie są uwzględniane cechy i ograniczenia docelowych procesorów. Wprowadza to dodatkowe spowolnienie w wykonywaniu kodu pośredniego, a niekiedy uniemożliwia nawet jego wykonanie, gdy różnice architektoniczne między maszyną wirtualną, a docelowym procesorem są znaczne.

Platformę badawczą w niniejszej pracy stanowi środowisko programistyczno-uruchomieniowe CPDev, które pracuje w oparciu koncepcję maszyny wirtualnej wykonującej kod pośredni. Główną zaletą systemu jest jego uniwersalność implementacyjna pozwalająca na umieszczenie go na dowolnej platformie wykonawczej, sprzętowej lub wirtualizowanej. Tym samym zdjęte zostaje ograniczanie wykorzystania sterownika programowanego w normie IEC 61131 tylko do specjalizowanych sterowników PLC. W związku z tym zadania dotyczące komunikacji, ustawień urządzeń wejść/wyjść oraz diagnostyki muszą być wykonywane niezależnie od kodu programu sterującego. W razie niepoprawnej pracy algorytmu możliwe jest bezpieczne zatrzymanie programu. Niezależnie od statusu programu sterownik może komunikować się z podłączonymi urządzeniami zapewniając bezpieczeństwo pracy systemu.

Na rysunku 1 przedstawiono uruchamianie programu CPYMEM_WORD w języku ST z listingu 1 w środowisku CPDev IDE (*Integrated Development Environment*). Jest na nim widoczny tryb zaawansowanego debugowania, w którym można śledzić wykonywanie kodu pośredniego przez maszynę wirtualną. Kod ten umieszczony jest w oknie po prawej stronie w formie binarnej (szesnastkowo) oraz z wyświetleniem mnemoników instrukcji maszyny wirtualnej (kolor niebieski). Spośród użytych instrukcji widoczne są: CALB (wywołanie podprogramu), TRML (zakończenie cyklu obliczeniowego), FPAT (inicjalizacja tablicy stałą wartością), MCD (inicjalizacja zmiennych), RETURN (powrót z podprogramu). Kod ten będzie dalej używany do badań wydajnościowych.



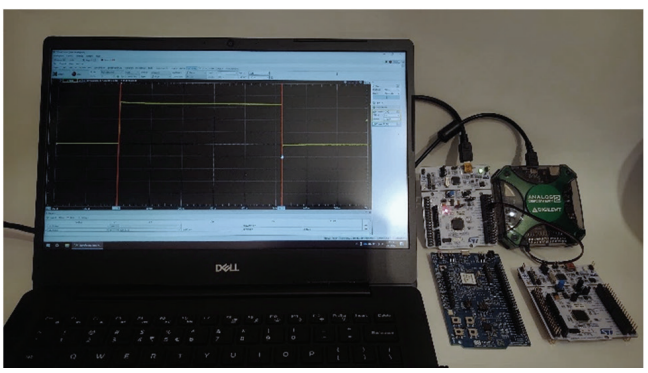
Rys. 1. Uruchamianie programu dla maszyny wirtualnej w środowisku CPDev
Fig. 1. Running the program for a virtual machine in the CPDev environment

4. Ograniczenia architektury ARM dla systemów wbudowanych

Pierwszy rdzeń ARM (ang. *Advanced RISC Machine*) o nazwie ARM1 został wprowadzony w 1985 r. i bazował na architekturze ARMv1. Przez następne lata opracowywane były kolejne rdzenie w oparciu o różne modele architektury, czego efektem jest występowanie wielu serii opartych niejednokrotnie na różnych architekturach. Zakres ten obejmuje wiele rodzin od ARM1 do ARM11, przez szeroko wykorzystywane Cortex w seriach M, A i R i najnowszą serwerową Neoverse. Aktualnie architektura ARM to jedna z najczęściej wykorzystywanych na świecie, stosowana w ponad 70% systemów wbudowanych [11]. Ponadto coraz chętniej używana jest do budowy klastrów obliczeniowych. Zmniejszona liczba instrukcji procesora przełożyła się na mniejsze skomplikowanie (liczba tranzystorów potrzebna do wykonania procesora) oraz zwiększoną efektywność energetyczną. Stosunkowo niska cena jednostkowa pozwoliła na masową implementacje procesorów w różnych układach scalonych.

W ostatnich latach, szczególnie popularne są rdzenie Cortex-M oraz Cortex-A [12, 13]. Seria M została zoptymalizowania pod kątem budowy mikrokontrolerów, a nacisk położony został na wysoką sprawność energetyczną i niski koszt produkcji. Występuje wyłącznie w wersji 32-bitowej, czyli słowa, adresy i dane mieszczą się na czterech bajtach pamięci. Stosowana jest w całym przekroju zadań, od niskowydajnych i ekonomicznych, po energooszczędne i najbardziej wydajne jednostki obliczeniowe do zadań specjalnych. Aktualnie grupa rdzeni Cortex-M składa się z rdzeni opartych na architekturze ARMv6-M, ARMv7E-M, ARMv8E-M oraz ARMv8.1-M. Z kolei rozbudowana seria Cortex-A zawiera procesory z wysokowydajnymi architekturami 32- i 64-bitowymi. Cechą szczególną tej serii jest zaimplementowana jednostka zarządzania pamięcią MMU (ang. *Memory Management Unit*). Jej przeznaczeniem jest zastosowanie w dużych systemach wykorzystujących systemy operacyjne.

Opracowanie niezawodnego, uniwersalnego i wydajnego oprogramowania wymaga dostosowania się do ograniczeń platformy. Zróżnicowanie procesorów utrudnia przygotowanie jednolitego, efektywnie działającego niskopoziomowego oprogramowania [14, 15]. Podczas implementacji maszyny wirtualnej na różnorodnych platformach dostrzeżono różnice związane z dostępem do pamięci. Architektura ARM wspiera pracę na wyrównanej adresacji. Rozumie się to jako operacje na pamięci, której adres jest podzielny przez bajtowy rozmiar danych. Dla operacji na 32 bitach adres powinien być podzielny przez cztery, a dla 16 bitów przez dwa. Nowsze, rozbudowane architektury procesorów wspierają dostęp niewyrównany, czyli np. na adresach nieparzystych [16].



Rys. 2. Stanowisko badawcze z pakietem pomiarowym Analog Discovery 2 firmy Digilent działającym w trybie oscyloskopu
Fig. 2. Research stand with the Digilent Analog Discovery 2 measurement package operating in oscilloscope mode

5. Testy wydajnościowe

Maszyna wirtualna CPDev występuje w trzech wersjach: 16- i 32-bitowej i 32-bitowej z wyrównaniem [4]. Zróżnicowanie rozmiaru i sposobu adresowania pozwala na implementacje maszyny na różnorodnych układach.

Dla układów wyposażonych w stosunkowo małą ilość pamięci programu i danych przewidziana jest maszyna 16-bitowa. Nakłada ona na nas ograniczenie adresowania, po 64 kB pamięci kodu i danych. W praktyce ilość ta jest wystarczająca do większości prostych systemów sterowania. Systemy wymagające znacznie większej liczby zmiennych lub bardziej rozbudowanych programów wykorzystują maszynę wirtualną w wersji 32-bitowej. Adresacja ta umożliwia zwiększenie rozmiaru kodu i danych do 4 GB. 32-bitowa konfiguracja maszyny wirtualnej wydaje się najbardziej odpowiednia do układów o architekturze ARM.

W celu przetestowania wydajności dostępu do pamięci poszczególnych rozwiązań układowych opracowany został kod testowy, którego zadaniem było wyeksponowanie informacji o różnicach czasu dostępu do pamięci, w zależności od rozmiaru stosowanych zmiennych w systemie sterowania. W tym celu każda z platform przetestowana została pod kątem 8-, 16-, 32- i 64-bitowych zmiennych typu BYTE, WORD, DWORD, LWORD w trzech różnorodnych trybach dostępu do pamięci danych.

Podstawowym trybem dostępu do pamięci przez maszynę wirtualną jest dostęp bajtowy. Jego zadaniem jest wyłuskanie potrzebnej ilości danych przy pomocy odczytu pojedynczych bajtów z pamięci, które finalnie składane są do wymaganego formatu półsłów, słów lub podwójnych słów. Metoda ta jest w pełni wspierana przez architekturę ARM, ze względu na to, że dostęp do bajtów jest zawsze wyrównany – brak ograniczeń.

Drugi tryb polega na kopiowaniu pamięci z wykorzystaniem standardowych funkcji języka C, której szczegółowy sposób dostępu do pamięci zależy od architektury, dla której kompilowana jest maszyna wirtualna. Zaproponowany tryb dostępu do pamięci jest uniwersalny i elastyczny pod względem przenaszalności na inne platformy.

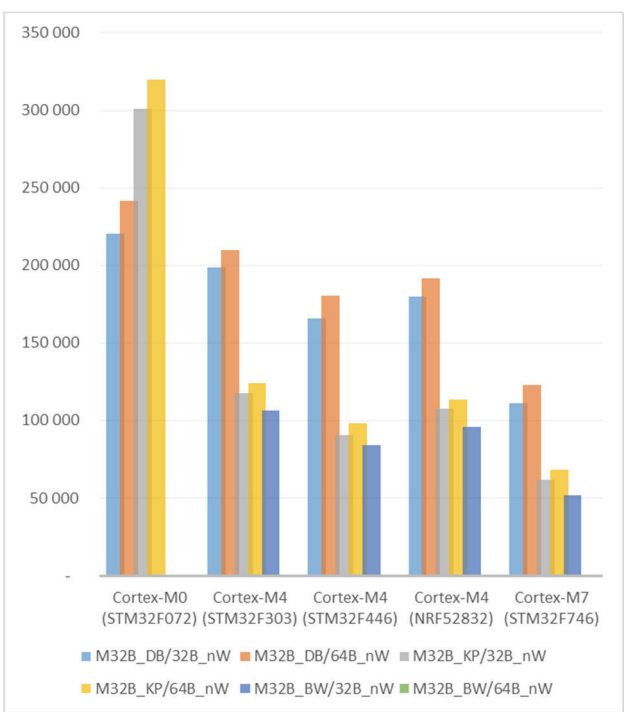
Trzeci rodzaj to dostęp bezpośredni przy pomocy wskaźników. Pozwala on na wykorzystanie potencjalnie najefektywniejszego sposobu odczytu lub zapisu za pomocą bezpośredniego odwołania do miejsca w pamięci. W przypadku omawianej architektury ARM, tryb ten nie jest wspierany w każdej z dostępnych serii procesorów. Problem ten szczególnie dotyczy ekonomicznych i starszych serii rdzeni nieposiadających instrukcji dostępu do pamięci o adresie niewyrównanym. Należy zaznaczyć, że poprawny dostęp do pamięci niewyrównanej może generować kary czasowe, które nie występują w trybie wyrównanym.

Wyniki poszczególnych badań efektywności i niezawodności obliczeniowej dla różnych platform przedstawione zostały na wykresach. Czas obliczeń na osi 0Y został znormalizowany i zapisany w mikrosekundach na MHz. Wykres ograniczono do 32- i 64- bitowych zmiennych (np. DWORD, LREAL, DATE_AND_TIME), dla mniejszych typów danych wyniki są analogiczne i dostępne na stronie projektu [17]. Do każdego z rysunków dla czytelności zaproponowana została skrócona legenda, w której występują poniższe oznaczenia:

- DB – dostęp bajtowy,
- KP – kopiowanie pamięci,
- BW – bezpośredni wskaźnik,
- nW – pamięć niewyrównana,
- W – pamięć wyrównana,
- M32B – maszyna wirtualna 32-bitowa.

Dla przykładu oznaczenie M32B_DB/64B_nW oznacza testy 64-bitowej zmiennej z pamięcią niewyrównaną na 32-bitowej maszyny wirtualnej z dostępem bajtowym.

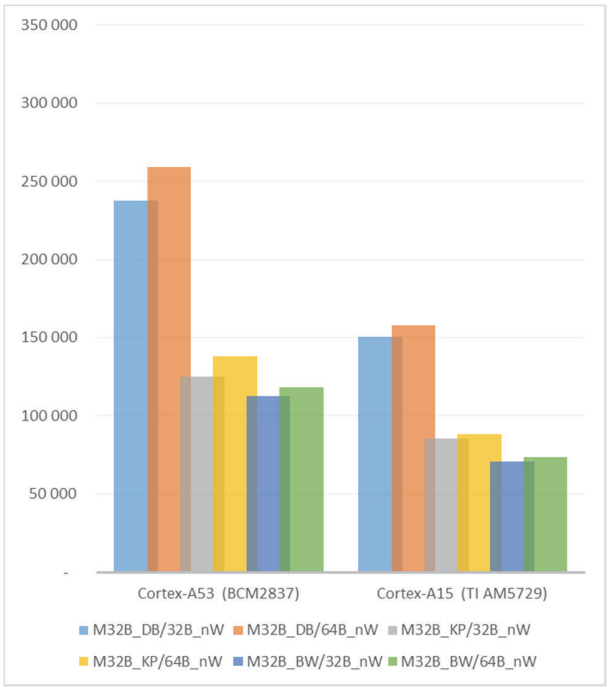
Maszyna wirtualna CPDev zaimplementowana i przetestowana została na pięciu różnych układach z rdzeniami Cortex-M, od najmniej wydajnego Cortex-M0, poprzez trzy wersje Cortex-M4, do najwyższej serii Cortex-M7. Znormalizowane wyniki (rys. 3) ukazują zauważalne różnice między platformami. Na szczególną uwagę zasługuje brak części wyników w każdej z przedstawionych platform. Najgorzej plasuje się układ z STM32F072 wykorzystujący Cortex-M0. Czas wykonywania cyklu programu był najdłuższy, ponadto dostęp bezpośredni w obydwu przypadkach (32- i 64-bitowa zmienna) zakończony został niepowodzeniem (wyjątek *hard fault*). Platformy wykorzystujące Cortex-M4 i Cortex-M7 nie poradziły sobie z dostępem wskaźnikowym dla zmiennych 64-bitowych. Dodatkowo, na uwagę zasługuje fakt, że przy porównaniu czasu wykonywaniu kodu dla jednego rozmiaru zmiennej, Cortex-M0 uzyskuje lepsze wyniki dla dostępu bajtowego niż dla kopiowania pamięci, czyli przeciwnie do reszty porównywanych platform.



Rys. 3. Czas wykonywania cyklu programu ze względu na rozmiar zmiennej dla różnych rodzajów dostępu pamięci na architekturze Cortex-M. Maszyna wirtualna CPDev w trybie 32-bitowym bez wyrównania

Fig. 3. Program cycle execution time due to the size of the variable for different types of memory access on the Cortex-M architecture. CPDev virtual machine in 32-bit mode with no alignment

Druga seria testów przeprowadzona została na układach z serii Cortex-A, której wyniki przedstawiano na rysunku 4. Platforma Cortex-A53 posiada średnio 31–39 % gorsze wyniki niż Cortex-A15. Na wykresie można dostrzec powtarzalność różnic wyników między różnorodnymi testami. W każdym z przypadków (niezależnie od metody dostępu do pamięci) czas wykonania cyklu dla zmiennych 64-bitowych uległ zwiększeniu. Ponadto najefektywniejszym sposobem dostępu jest odwołanie wskaźnikowe, natomiast kopiowanie pamięci wprowa-



Rys. 4. Czas wykonywania cyklu programu ze względu na rozmiar zmiennej dla różnych rodzajów dostępu pamięci na architekturze Cortex-A. Maszyna wirtualna CPDev w trybie 32-bitowym bez wyrównania

Fig. 4. Program cycle execution time due to the size of the variable for different types of memory access on the Cortex-A architecture. CPDev virtual machine in 32-bit mode with no alignment

dza niewielki wzrost czasu wykonywania cyklu. Najmniej efektywnym sposobem jest dostęp bajtowy, który jest średnio dwa razy dłuższy niż alternatywne sposoby.

Na podstawie przeprowadzonych badań można wysnuć poniższe wnioski. Jeżeli wybrana platforma wspiera dostęp niewyrównany to najbardziej wydajnym sposobem dostępu do pamięci jest dostęp bezpośredni. Nieświadome wykorzystanie tej metody może zakończyć się niepoprawnym działaniem lub zablokowaniem procesora. Uniwersalną, a zarazem efektywną metodą jest kopiowanie pamięci. Metoda ta działa poprawnie na każdej z testowanych platform, zmniejszając czas wykonywania cyklu programu o kilkanaście procent. Z wyjątkiem niskowydajnych rdzeni takich jak Cortex-M0, najmniej efektywną metodą jest dostęp bajtowy. Na podstawie wyników dla Cortex-M i Cortex-A można dostrzec problem dostępu bezpośredniego, który dotyczy całej serii M przeznaczonej dla mikrokontrolerów.

6. Proponowane usprawnienia

W celu zminimalizowania wpływu ograniczeń dostępu do pamięci w architekturze ARM zdecydowano się rozszerzyć specyfikację maszyny wirtualnej oraz odpowiednio zmodyfikować działanie kompilatora CPDev. Zmiany dotyczyły wyrównania danych i objęły następujące obszary:

- generowanie kodu pośredniego z rozkazami wyrównanymi do 32 bitów (4 bajtów) w pamięci kodu,
- wprowadzenie nowych wariantów instrukcji przetwarzających dane wyrównane do 32 bitów.

Ze względu na architekturę harwardzką maszyny wirtualnej CPDev, wyrównanie dotyczy adresów (wskaźników) w pamięci kodu programu i pamięci danych, gdzie przechowywane są zmienne.

Dzięki odpowiedniej formie kodu pośredniego, maszyna zawsze pobiera rozkaz z adresów dostępnych dla procesora ARM bezpośrednio, tj. wyrównanych do 32 bitów. Wszystkie operandy są również zorganizowane tak, aby dało się je pobrać z wyrównanych adresów pamięci kodu.

Nowe warianty instrukcji dotyczą procedur systemowych obsługujących pamięć danych i kodu, oznaczone końcówką 4A (ang. *4 byte Alignment*) zebrano w Tabeli 2. Dwa ostatnie rozkazy związane są z możliwością wywołania zewnętrznego kodu przygotowanego w formie tzw. bloków natywnych, czyli skompilowanych dla określonego procesora i tworzonych najczęściej w języku C [17].

Warto dodać, że podczas kompilacji moduł zwany generatorem kodu pośredniego ustala adresy zmiennych w pamięci danych tak, aby były podzielne przez cztery, czyli z wyrównaniem. Daje to możliwość usprawnienia ich odczytu i zapisu w architekturze z ograniczeniami dostępu do pamięci.

Na listingu 3 przedstawiono fragment kodu dla maszyny wirtualnej przygotowanego z użyciem nowego rozwiązania. Pierwsza część (od etykiety `:?CpyMem.CPYMEM_BYTE?INIT`) używa instrukcji FPAT4A i MCD4A w celu zainicjowania tablic BUF1 i BUF2 oraz zmiennej I. Widoczne jest nadmiarowe wypełnienie obszaru danych wartościami 0xFF w celu uzyskania wyrównania. W drugiej części (od etykiety `:?CpyMem.CPYMEM_BYTEFOR?ADD`) przy dodawaniu stosowana jest instrukcja MEMCP4A, pełniąca rolę przypisania z wyrównaniem.

```
  :00000020| :?CpyMem.CPYMEM_BYTE?INIT
  :00000020| $VMSYS.FPAT4A BUF1, #0000, #1400
  :0000002c| $VMSYS.FPAT4A BUF2, #0000, #1400
  :00000038| $VMSYS.MCD4A I, #02000000, #0000FFFFFFFF
  ...
  :00000088| $VMSYS.RETURN
  ...
  :0000009c| :?CpyMem.CPYMEM_BYTEFOR?ADD
  ...
  ; buf1[i] := (buf2[i] + BYTE#3);
  :000000b8|  $VMSYS.MEMCP4A  ?LAC?BUF20034,  I,
#02000000000
  :000000c8| $VMSYS.CEAC ?LAC?BUF20034, ?L?CST0035,
?L?CST0030
  :000000d8|  $VMSYS.AORD  ?LRDA?BUF20036,  BUF2,
?LAC?BUF20034
  :000000e8| $DEFAULT.ADD ?TEMP?0032, ?LRDA?BUF20036,
?L?CST0033
  :000000f8|  $VMSYS.MEMCP4A  ?LAC?BUF10038,  I,
#02000000000
  :00000108| $VMSYS.CEAC ?LAC?BUF10038, ?L?CST0035,
?L?CST0030
  :00000118| $VMSYS.AOWR ?TEMP?0032, BUF1, ?LAC?
BUF10038
  ...
  :000001b0| $VMSYS.RETURN
```

Listing 3. Kod pośredni dla maszyny wirtualnej uwzględniający nowe procedury

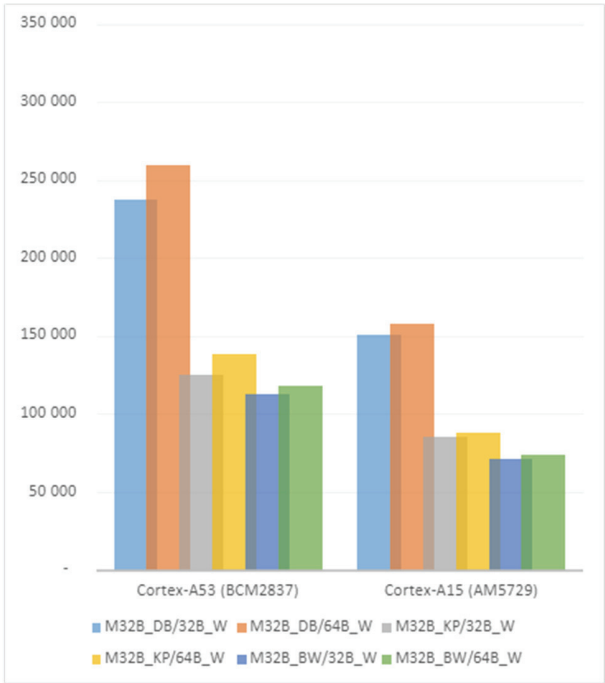
Listing 3. Intermediate code for the virtual machine with new procedures

7. Rezultaty

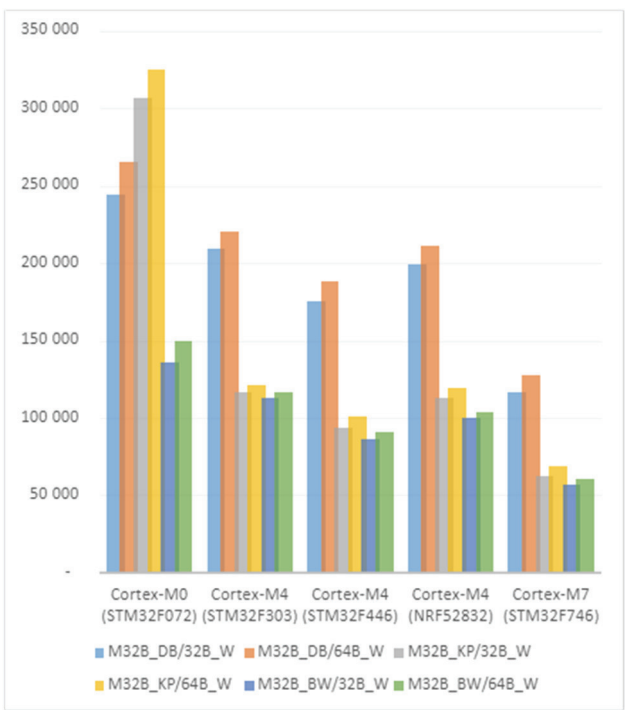
Po wprowadzeniu modyfikacji kompilatora i maszyny wirtualnej dostosowującej ją do charakterystyki pracy architektury ARM, z adresacją wyrównaną do pełnego słowa i nowymi instrukcjami maszyny wirtualnej ponownie przeprowadzona

Tabela 2. Nowe procedury systemowe
Table 2. New system procedures

Nazwa	Argumenty	Opis
MCD4A	*dst, size, pattern*	Inicjalizacja pamięci danych od adresu *dst* danymi *pattern* o rozmiarze *size* (wyrównanie do 4B za pomocą 0xFF)
MEMCP4A	*dst, src, count*	Kopiuje obszar pamięci danych z *src* do *dst* o rozmiarze *count* z wyrównaniem
FPAT4A	*dst, val, count*	Wypełnia obszar *dst* o rozmiarze *count* wartością *val* z wyrównaniem
DPRDL4A	*var, src, count*	Kopiuje dane z *src* o rozmiarze *count* do zmiennej *var*
DPWRL4A	*var, dst, count*	Kopiuje wartość zmiennej *var* do obszaru *dst* o rozmiarze *count*
HWFBI4A HWFBC4A	*id, instance*	Inicjalizacja i wywołanie instancji *instance* bloku natywnego typu *id*



Rys. 5. Czas wykonywania kodu programu ze względu na rozmiar zmiennej dla różnych rodzajów dostępu pamięci na architekturze Cortex-M. Maszyna wirtualna CPDev w trybie 32-bitowym z wyrównaniem
Fig. 5. Program cycle execution time due to the size of the variable for different types of memory access on the Cortex-M architecture. CPDev virtual machine in 32-bit mode with alignment



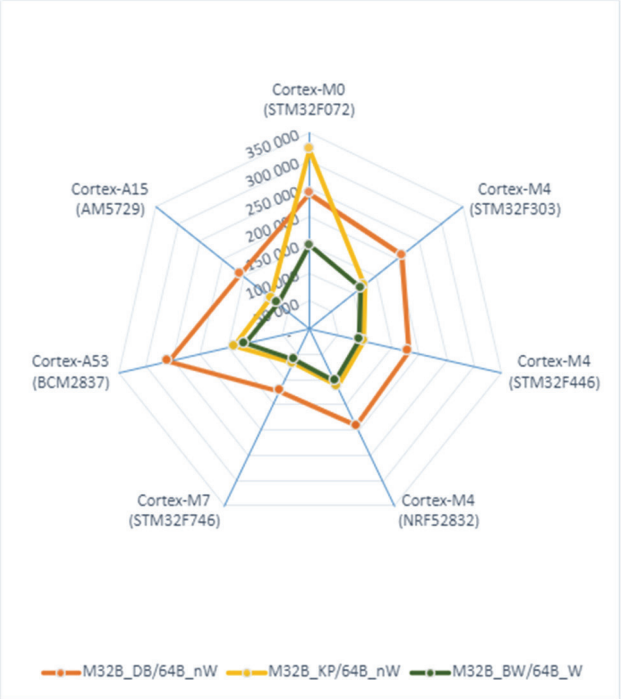
Rys. 6. Czas wykonywania kodu programu ze względu na rozmiar zmiennej dla różnych rodzajów dostępu pamięci na architekturze Cortex-A. Maszyna wirtualna CPDev w trybie 32-bitowym z wyrównaniem
Fig. 6. Program cycle execution time due to the size of the variable for different types of memory access on the Cortex-A architecture. CPDev virtual machine in 32-bit mode with alignment

została seria testów porównawczych.

Wyniki dla platformy Cortex-M przedstawione zostały na rysunku 5. Dzięki wprowadzonemu rozszerzeniu 32-bitowego wyrównania instrukcji i danych maszyny wirtualnej uzyskano planowaną kompatybilność dostępu wskaźnikowego niezależnie od rozmiaru odczytywanej zmiennej. Metoda bezpośredniego wskaźnika poprawiła wyniki czasu obliczeń w przybliżeniu dwukrotnie. Dla serii Cortex-M4 i Cortex-M7 jest też o kilka procent efektywniejsza względem metody kopiowania pamięci. Dla najniższej serii w celu optymalizacji czasu obliczeń zaleca się stosowanie metody dostępu bajtowego.

Czasy cykli obliczeniowych przeprowadzone na platformie Cortex-A przedstawione zostały na rysunku 6. W przypadku serii mikroprocesorowej różnice między dwoma metodami dostępu mieszczą się w zakresie błędu pomiarowego wykorzystanych do tego celu metod. Wskazuje to na bezkompromisową budowę architektury, której instrukcje wspierają pracę z dowolnie rozmieszczonymi danymi.

Na podstawie przeprowadzonych badań wytypowane zostały trzy metody poprawnego dostępu do pamięci w architekturze ARM. Metoda dostępu bajtowego oraz kopiowania pamięci może być z powodzeniem stosowana przy dowolnym rozmieszczeniu danych w pamięci. W celu zminimalizowania czasu dostępu do



Rys. 7. Porównanie czasu wykonywania cyklu programu dla zmiennych 64-bitowych na platformach Cortex-M i Cortex-A w trzech efektywnych metodach dostępu do pamięci
Fig. 7. Comparing program cycle runtime for 64-bit variables on Cortex-M and Cortex-A platforms in three effective memory access methods

pamięci należy stosować ułożenie danych wyrównanych do pełnego słowa. Na rysunku 7 porównano efektywne metody dostępu do pamięci na platformie Cortex-M i Cortex-A. Z wyjątkiem najmniej wydajnego Cortex-M0, zaleca się stosowanie metody kopiowania pamięci i bezpośredniego wskaźnika. W przypadku najniższej serii Cortex-M, dostęp bajtowy dla niewyrównanych danych jest efektywniejszy niż metoda kopiowania pamięci.

8. Podsumowanie

Systemy sterowania często wykorzystują układy mikroprocesorowe o architekturze ARM, lecz ograniczenia dostępu do pamięci nie są w wielu przypadkach brane pod uwagę przez twórców ze względu na optymalizacje wykonywane automatycznie przez kompilatory języków programowania. Inaczej jest w przypadku kodu pośredniego dla maszyn wirtualnych, który nie musi uwzględniać charakterystyki docelowych platform sprzętowych.

Wyniki badań pokazały istotny wpływ ograniczeń ARM na działanie maszyny wirtualnej w zakresie przetwarzania danych. Co więcej, wpływ ten zależy od typu rdzenia ARM, jego przeznaczenia, a nawet producenta układu.

Zaproponowane w artykule usprawnienia rozszerzające maszynę wirtualną o nowe instrukcje i wprowadzenie wyrównania do 32 bitów pozwalają uzyskać dobre rezultaty dla wszystkich badanych platform. W niektórych z nich dostęp do danych typu LWORD, LREAL, DATE_AND_TIME (64 bity) musi być realizowany przez kopiowanie pamięci, a nie poprzez bezpośrednie wskazanie, ale można to osiągnąć przez odpowiednie skonfigurowanie maszyny wirtualnej. Wadą proponowanego rozwiązania jest zwiększenie rozmiaru pamięci potrzebnej do przechowywania kodu i danych przez konieczność uwzględnienia wyrównania.

Podziękowania

Projekt finansowany w ramach programu Ministra Edukacji i Nauki pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019–2022 nr projektu 027/RID/2018/19 kwota finansowania 11 999 900 zł.

Bibliografia

1. International Electrotechnical Commission, "PN-EN 611313:2013 – Programmable controllers – Part 3: Programming languages," European Committee for Electrotechnical Standardization, Tech. Rep, 2013.
2. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Developing a multiplatform control environment*, Journal of Automation, Mobile Robotics and Intelligent Systems, Vol. 13, No. 4, 2019, 73–84, DOI: 10.14313/JAMRIS/ 4-2019/40.
3. Trybus B., *Development and Implementation of IEC 61131-3 Virtual Machine*, "Theoretical and Applied Informatics", Vol. 23, No. 1, 2011, 21–35.
4. Hubacz M., Sadolewski J., Trybus B., Wydajność architektury STM32 w zakresie wykonywania kodu pośredniego dla systemów sterowania, „Pomiary Automatyka Robotyka", R. 25, Nr 1, 2021, 27–34, DOI: 10.14313/PAR_239/27.
5. Kim H.S., Lee J.Y., Kwon W.H., *A compiler design for IEC 1131-3 standard languages of programmable logic controllers*, [in:] SICE '99 Ann. Conf., 1999, 1155–1160, DOI: 10.1109/SICE.1999.788715.
6. LLVM Compiler Infrastructure. [www.llvm.org]
7. Rockwell Automation, ISaGRAF Workbench. [www.isagraf.com]
8. COPA-DATA France, STRATON. [www.straton-plc.com]
9. Zhou C., Chen H., *Development of a PLC Virtual Machine Orienting IEC 61131-3 Standard*, [in:] International Conference on Measuring Technology and Mechatronics Automation, Vol. 3, 2009, 374–379, DOI: 10.1109/ICMTMA.2009.422.
10. Zhang M., Lu Y., Xia T., *The Design and Implementation of Virtual Machine System in Embedded SoftPLC System*, [in:] International Conference on Computer Sciences and Applications, 2013, 775–778, DOI: 10.1109/CSA.2013.185.
11. *Introducing Cortex-A32: ARM's smallest, lowest power ARMv8-A processor*, https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/introducing-cortex-a32-arm-s-smallest-lowest-power-armv8-a-processor-for-next-generation-32-bit-embedded-applications

12. *Arm Limited, Q2 2018 Roadshow Slides*, https://www.arm.com/-/media/global/company/investors/Financial%20Result%20Docs/Arm_SB_Q2_2018_Roadshow_Slides_FINAL.pdf?revision=b8ffe77d-e34f-4dc1-b0ec-69726f536cc6
13. *ARM Microcontrollers Market Size And Forecast*, https://www.verifiedmarketresearch.com/product/arm-microcontrollers-market/
14. Wang K.C., *Embedded and Real-Time Operating Systems*, 2017, 401–475, DOI: 10.1007/978-3-319-51517-5_10.
15. *Dokumentacja architektury ARM*, https://developer.arm.com/documentation/
16. Kusswurm D., *Modern Arm Assembly Language Programming*, ISBN: 978-1-4842-6266-5, Apress, Berkeley, CA, 2020.
17. CPDev VM public sources. https: //github.com/CPDev-ControlProgramDeveloper

Support for PN-EN 61131-3 Standard Data Types in ARM Architecture with Memory Access Restrictions

Abstract: The article presents the results of research on the handling of data types from the PN-EN 61131-3 standard in systems with ARM architecture. The tests were carried out on several different hardware platforms with the Cortex-M and Cortex-A series as CPUs. The research was carried out on the basis of the CPDev environment for creating and running control software. Due to the limitations of the ARM architecture, three methods of access to memory have been developed, and the results allow to determine the most effective. The article also presents the proposed virtual machine extension with new instructions to make data operations in ARM solutions more efficient.

Keywords: PLC, ARM, PN-EN 61131-3, CPDev

mgr inż. Marcin Hubacz
m.hubacz@prz.edu.pl
ORCID: 0000-0002-2748-11454

W 2019 r. ukończył studia na Wydziale Elektrotechniki i Informatyki Politechniki Rzeszowskiej – kierunek Automatyka i Robotyka oraz Informatyka. Obecnie Asystent w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego główne zainteresowania dotyczą robotyki, elektroniki, systemów wbudowanych oraz druku 3D.

dr inż. Jan Sadolewski
jsad@prz.edu.pl
ORCID: 0000-0001-7370-9027

Absolwent Wydziału Elektrotechniki i Informatyki Politechniki Rzeszowskiej (2006 r.). W 2012 r. uzyskał stopień doktora nauk technicznych w dyscyplinie informatyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej w Gliwicach. Jego zainteresowania naukowe koncentrują się wokół języków programowania, tworzenia kompilatorów oraz środowisk wykonawczych.

dr inż. Bartosz Trybus
btrybus@kia.prz.edu.pl
ORCID: 0000-0002-4588-3973

Adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Ukończył studia na Wydziale Elektrycznym, Automatyki, Informatyki i Elektroniki AGH w Krakowie. Doktorat z informatyki uzyskał w 2004 r. Jego główne badania dotyczą systemów czasu rzeczywistego i środowisk wykonawczych oprogramowania sterującego.