Metody realizacji operacji bitowych języka LD w urządzeniach sterujących opartych o architekturę AVR

Łukasz Herb

1. Wprowadzenie

Nowoczesne systemy sterowania maszyn i urządzeń coraz rzadziej budowane są w oparciu o logikę przekaźnikowo-stycznikową. Rolę elementów wykonawczych i sterujących przejmują urządzenia mikroprocesorowe, takie jak sterowniki PLC. Realizują one algorytm sterowania, przedstawiony w formie programu dla mikroprocesora, pobierają dane z podłączonych cyfrowych i analogowych czujników oraz sterują dyskretnymi i wielostanowymi wyjściami. Elementy czujnikowe, wykonawcze oraz sterujące mogą zostać rozproszone po całym obiekcie, którym sterują, ale dzięki sieciom przemysłowym mogą w deterministycznym czasie wymieniać między sobą potrzebne im dane.

Podstawowym czujnikiem podłączonym do sieci przemysłowej jest wejście dwustanowe dostarczające dane o wielkości jednego bitu. Wyłączniki krańcowe, przełączniki, kurtyny bezpieczeństwa itp. udostępniają do urządzenia akwizycji wejść tylko informację o zadziałaniu lub nie. Dzięki rozmiarowi takiej informacji stan wielu czujników może być przechowywany w jednym słowie i przekazany przez sieć jednocześnie.

Z drugiej strony, najprostsze urządzenia wyjściowe z systemu także realizują tylko dwa stany pracy: otwarcie lub zamknięcie zaworu hydraulicznego, załączenie sygnalizatora akustycznego lub wysterowanie stycznika i załączenie silnika.

Przedstawiony w artykule język drabinkowy LD (ang. *Ladder Diagram*) posiada pulę specjalnych rozkazów operujących na pojedynczych bitach, reprezentujących stan czujników binarnych, a także pozwala na sterowanie dwustanowymi wyjściami elementów wykonawczych.

W artykule przedstawiono metodę realizacji operacji logicznych w systemie mikroprocesorowym o architekturze AVR. Dane, będące argumentami operacji, pobierane są poprzez sieć polową, np. Modbus z urządzenia akwizycji wejść dyskretnych – dwustanowych.

2. Język drabinkowy

Wzorowany na schematach elektrycznych stykowo-przekaźnikowych systemów sterowania język drabinkowy LD (ang. *Ladder Diagram*) został ujęty w normie IEC 61131-3 opisującej tekstowe i graficzne języki programowania sterowników PLC [1].

W normie [2] zdefiniowano także elementarne typy danych, z których dla tego artykułu ważny jest jedynie typ BOOL. Zmienna tego typu ma wielkość jednego bitu i przyjmować może dwa stany FALSE (fałsz) oraz TRUE (prawda) odpowiadające wartościom liczbowym 0 oraz 1.

Programy w języku LD zbudowane są ze zbiorów połączonych ze sobą elementów graficznych grupowanych w obwody (rys. 1 a). Następnie, aby wskazać sekwencyjne wykonanie operacji, obwody umieszczane są względem siebie pionowo, tworząc szczeble (rys. 1 b). W ramach jednego obwodu algorytm sterowania realizowany jest zgodnie z przepływem prądu (ang. *power flow*) od lewej szyny zasilania, sprawdzając wszystkie warunki aż do prawej strony (rys. 1 c).

Podstawowymi elementami języka LD są styki oraz cewki. Reprezentują one dwustanowe wejścia oraz wyjścia.

Streszczenie: W niniejszym artykule przedstawiono problem realizacji operacji logicznych, będącymi podstawowym elementem języka drabinkowego LD. Pokazano metody dostępu do danych jednobitowych, sposoby wykonania operacji na nich oraz cechy architektury systemu, które wspomagają oraz utrudniają te operacje. Omawiane urządzenia zbudowane są jako systemy mikroprocesorowe o ośmiobitowej architekturze AVR.

Słowa kluczowe: operacje logiczne, mikroprocesory, AVR.

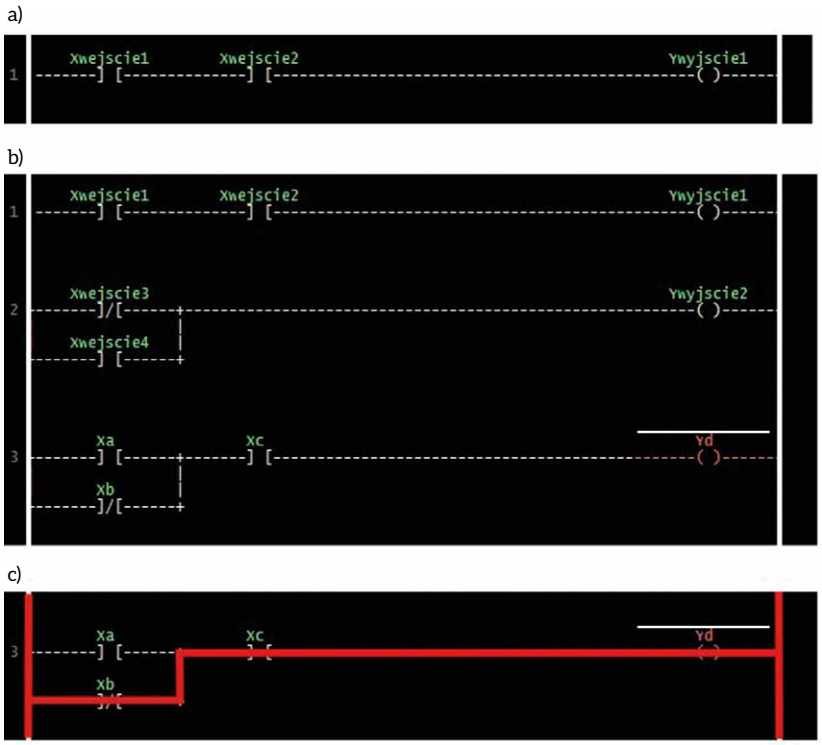
THE METHODS OF THE BINARY OPERATIONS IN CONTROL DEVICES BASED ON AVR ARCHITECTURE

Abstract: *This paper presents problem of implementation of logical operations, that are the basic element of LD language. It shows methods of access to a single-bit data, ways of executing operations on the data and features of system architecture that support or complicate this operation. The described control devices that were built as microprocessor systems with 8 bit AVR architecture.*

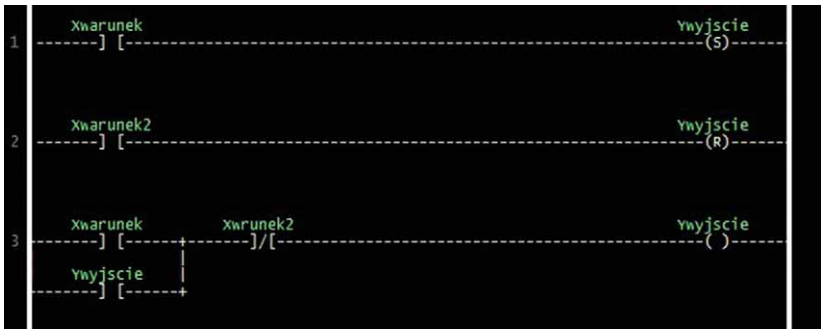
Keywords: logical operations, microprocessors, AVR.

W tabeli 1 przedstawiono rodzaje cewek i styków, które zostaną omówione w tym artykule.

Zestaw symboli dodatkowo można zredukować, zauważając, że cewki zatrzaskiwane to odpowiednie połączenie



Rys. 1. a) Jeden obwód; b) trzy szczeble; c) sterowanie przepływem prądu w warunkach: xa = 0, xb = 0, xc = 1. Zrzut ekranu z aplikacji LDmicro



Rys. 2. Realizacja cewek zatrzaskiwanych: 1) ustawianie; 2) kasowanie; 3) jednoczesna realizacja gałęzi 1 i 2

Tabela 1. Podstawowe symbole graficzne języka LD omawiane w artykule

Rodzaj	Symbol	Funkcja
Styk statyczny normalnie otwarty	`wejście -[ ]-`	Stan po lewej stronie symbolu przenoszony jest na prawą stronę, jeżeli wejście jest w stanie logicznym 1. W przeciwnym przypadku po prawej stronie pojawia się stan 0.
Styk statyczny normalnie zamknięty	`wejście -[/]-`	Stan po lewej stronie symbolu przenoszony jest na prawą stronę, jeżeli wejście jest w stanie logicznym 0. W przeciwnym przypadku po prawej stronie pojawia się stan 0.
Cewka zwykła	`wyjście -( )-`	Stan po lewej stronie symbolu przepisywany jest do zmiennej wyjście.
Cewka negująca	`wyjście -(/)-`	Stan przeciwny do stanu po lewej stronie symbolu przepisywany jest do zmiennej wyjście.
Cewka zatrzaskiwana ustawiająca	`wyjście -(S)-`	Jeżeli stan po lewej stronie symbolu ma wartość logiczną 1, to zmienna wyjściowa ustawiana jest w stan 1.
Cewka zatrzaskiwana kasująca	`wyjście -(R)-`	Jeżeli stan po lewej stronie symbolu ma wartość logiczną 1, to zmienna wyjściowa ustawiana jest w stan 0.

pozostałych symboli, co przedstawiono na rys. 2. W pierwszym i drugim obwodzie bit wyjściowy jest ustawiany i kasowany w zależności od warunków. Trzeci obwód realizuje to samo zadanie za pomocą zwykłej cewki i logiki ze sprzężeniem zwrotnym.

3. Architektura AVR

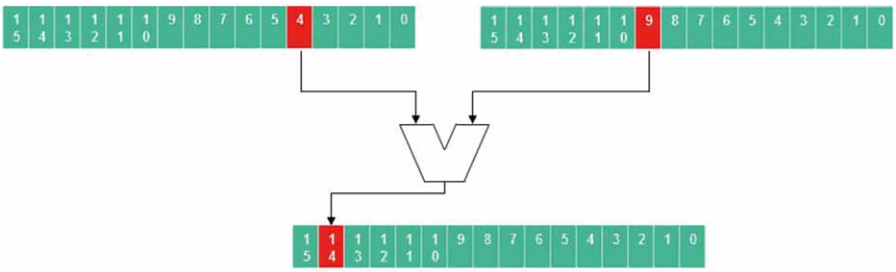
Atmel AVR to rodzina ośmiobitowych mikrokontrolerów o architekturze harvardzkiej (rozdzielona pamięć programu i danych). Lista rozkazów mikroprocesorów AVR zbudowana jest w oparciu o schemat RISC (ang. *Reduced Instruction Set Computing* – zredukowana lista instrukcji), co oznacza, że posiada tylko niezbędne minimum instrukcji, a jednostka arytmetyczno-logiczna operacje wykonuje jedynie na rejestrach ogólnego przeznaczenia. Do rdzenia mikroprocesora podłączono 32 rejestry, a wymiana danych między nimi a pamięcią odbywa się z wykorzystaniem rozkazów zapisz (ang. *store*) i załaduj (ang. *load*). Ważną cechą architektury AVR jest dwustopniowy potok rozkazów [4], dzięki czemu w tym samym czasie wykonywane są fragmenty dwóch rozkazów, powoduje to jednak problemy podczas analizy czasu wykonania programu zawierającego skoki warunkowe.

4. Sposoby realizacji rozkazów bitowych

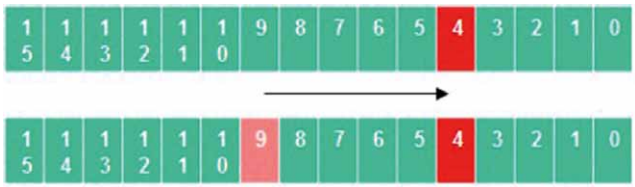
Podstawowym typem danych w języku drabinkowym, opisanym w akapicie 2, jest BOOL, czyli zmienna logiczna – bitowa. Za pomocą odpowiedniego łączenia styków i cewek realizować można operacje logicznej sumy (połączenie równoległe) i iloczynu (połączenie szeregowe), których tablice prawdy przedstawiono w tabeli 2.

Tabela. 2. Tablice prawdy dla operacji sumy i iloczynu bitowego

Wartość pierwszego argumentu	Wartość drugiego argumentu	Wartość iloczynu	Wartość sumy
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



Rys. 3. Schemat operacji bitowej w systemie 16-bitowym



Rys. 4. Wyrównanie bitowych pozycji argumentów

Ponieważ omawiane mikroprocesory należą do rodziny 8-bitowej, najmniejszą jednostką organizacji danych, na której mogą przeprowadzać operacje, jest bajt zawierający osiem bitów. W kolejnych paragrafach przedstawione zostaną algorytmy realizacji operacji bitowych w architekturach większych niż jednobitowe.

4.1. Algorytm używający operacji logicznych

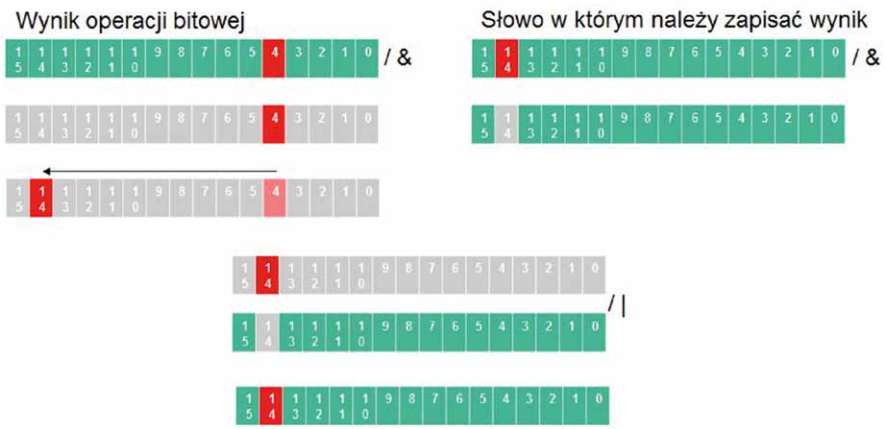
Dostępne w mikroprocesorach instrukcje logiczne najczęściej operują na wszystkich bitach słowa jednocześnie. W ogólnym przypadku należy wykonać operację na dwóch bitach znajdujących się w różnych bajtach pamięci, na różnych pozycjach, a wynik zapisać w innym bajcie na jeszcze innej pozycji (rys. 3). W tym celu zaproponowano następujące algorytmy, zakładając architekturę RISC – operacje logiczne wykonywane w modelu rejestr – rejestr.

Metoda operująca na rejestrach:

1. Pobranie z pamięci bajtu zawierającego bit pierwszego argumentu.
2. Przesunięcie bitowe w lewo lub w prawo, aby umieścić bit argumentu na założonej pozycji (rys. 4).
3. Pobranie z pamięci drugiego bajtu zawierającego bit drugiego argumentu.
4. Przesunięcie bitowe rejestru z drugim argumentem tak, aby znalazł się na tej samej pozycji, co pierwszy argument.
5. Wykonanie operacji logicznej na obu bajtach.

W metodzie nieoperującej bezpośrednio na pamięci wynik operacji pozostaje w rejestrach ogólnego przeznaczenia, aby zapisać go w odpowiednim miejscu bajtu wynikowego zaproponowano następujący algorytm (rys. 5):

1. Pobranie bajtu, do którego należy zapisać bit wyniku operacji.
2. Operacja logicznego iloczynu w celu wyzerowania bitu wyniku w bajcie docelowym.
3. Przesunięcie bitowe wyniku operacji logicznej (jeżeli jest jeszcze konieczne), aby wyrównać pozycję wyniku z bajtem docelowym.
4. Operacja logicznego iloczynu w celu wyzerowania nieznaczących bitów w bajcie zawierającym wynik operacji logicznej.
5. Operacja sumy logicznej, zapisująca wynik operacji na odpowiedni bit bajtu docelowego.
6. Zapisanie bajtu z wynikiem do pamięci.



Rys. 5. Zapis bitu wynikowego do właściwego słowa w pamięci

4.2. Algorytm używający operacji arytmetycznych

Operacje logiczne zrealizować można także za pomocą elementów arytmetycznych, jakim jest np. sumator jednobitowy [6]. W systemach mikroprocesorowych realizacje logicznego iloczynu i sumy można przedstawić jako operacje arytmetyczne na liczbach w zakresie od 0 do 1 z nasyceniem. Ponieważ w wielu architekturach mikroprocesorowych operacje mnożenia i dodawania są dostępne, przedstawiono propozycję algorytmu realizacji zagadnienia w ten sposób.

Suma i iloczyn logiczny:

1. Pobranie z pamięci bajtu zawierającego bit pierwszego argumentu.
2. Operacja logicznego iloczynu w celu wyzerowania nieznaczących bitów w bajcie zawierającym pierwszy argument.
3. Pobranie z pamięci bajtu zawierającego bit drugiego argumentu.
4. Operacja logicznego iloczynu w celu wyzerowania nieznaczących bitów w bajcie zawierającym drugi argument.
5. Operacja arytmetycznej sumy lub arytmetycznego iloczynu obu bajtów.
6. Wynik przechowywany jako stan zero akumulatora.

Wynik operacji przeprowadzonej według powyższego algorytmu nie jest już pojedynczym bitem w rejestrze ogólnego przeznaczenia, ale jest reprezentowany, w zależności od architektury, jako flaga operacji arytmetycznej, której wynikiem było 0 lub jako zero w rejestrze docelowym.

5. Realizacja operacji logicznych w architekturze AVR

Ze względu na organizację rdzenia mikroprocesora o architekturze AVR wyróżniono warte uwagi elementy przedstawionych algorytmów:

1. Metoda wyrównywania pozycji bitów w operandach.
2. Realizacja operacji logicznych z wykorzystaniem charakterystycznych cech architektury AVR.

Dla każdej z realizacji algorytmu przedstawiono liczbę cykli procesora potrzebną do jego wykonania.

5.1. Realizacja za pomocą operacji logicznych

Algorytm przedstawiony w punkcie 4.1. można wprost zrealizować tak, jak pokazano to w tabeli 3. Oba argumenty muszą zostać pobrane z pamięci do rejestrów za pomocą instrukcji LD. Następnie pozycje bitów obu operandów są wyrównywane i wykonywana jest właściwa operacja logiczna na obu rejestrach. Założono, że numer bitu argumentu pierwszego jest większy niż argumentu drugiego

Ze względu na dostępność jedynie operacji przesunięcia bitowego w lewo i w prawo, a także zamiany tetrad miejscami obliczono liczbę cykli procesora potrzebnych na przesunięcie bitu na zadaną pozycję. W tabeli 4 przedstawiono potrzebną liczbę cykli do przesunięcia

bitu z pozycji *r* na pozycję *d*. Zauważając, że w architekturze AVR dostępny jest rozkaz nie tylko przesunięcia, ale i obrotu rejestru przez bit przeniesienia C, dokonano podobnej analizy dla tych rozkazów, a wyniki ujęto w tabeli 5.

5.2. Realizacja za pomocą operacji arytmetycznych

Na liście rozkazów mikrokontrolerów AVR znajdują się operacje arytmetyczne dodawania i mnożenia [3] pozwalające zrealizować algorytm 4.2.

W tym przypadku wynik operacji przechowywany jest jako binarny stan flagi Z. Flaga ta jest ustawiona na 1 automatycznie, gdy w wyniku operacji arytmetycznej obliczone zostanie zero. Zatem wynikiem operacji bitowej jest stan przeciwny do Z.

5.3. Realizacja z wykorzystaniem cech architektury AVR

Oprócz rozkazów operujących na całych rejestrach, w puli instrukcji AVR [3] znajdują się operacje testowania stanu konkretnego bitu w rejestrze oraz flaga T służąca do przechowywania wartości binarnej. Z użyciem tych rozkazów operacje logicznego iloczynu pokazano w tabeli 7.

Zastosowanie skoku warunkowego do sterowania przepływem programu powoduje opróżnienie potoku w przypadku błędu predykcji. W architekturze AVR algorytm predykcji skoku zakłada, że zawsze się on nie wykona [4]. Z tego względu rozkaz SBRS zajmuje jeden cykl, gdy warunek nie jest spełniony i kolejna instrukcja zostanie wykonana. Gdy następuje błąd predykcji, a zatem opróżnienie potoku, rozkaz ten jest wykonywany przez 2 cykle. Takie działanie umożliwi zachowanie stałej liczby cykli wykonania algorytmu niezależnie od wartości zmiennych wejściowych.

5.4. Zapis wyniku

Przedstawione wcześniej realizacje algorytmów dla procesorów o architekturze AVR swoje wyniki zwracają na trzy sposoby:

1. Na konkretnym bicie rejestru ogólnego przeznaczenia.
2. Poprzez stan wyniku operacji pamiętany we fladze Z.
3. Poprzez programowo ustawianą flagę T.

Tabela 3. Realizacja algorytmu 4.1. w AVR

Rozkaz	Argumenty	Liczba cykli	Znaczenie
LD	R2,k1	2	Pobranie pierwszego argumentu z pamięci (z adresu k1 do rejestru R2).
LD	R0,k2	2	Pobranie drugiego argumentu z pamięci (z adresu k2 do rejestru R0).
SWAP	R0	1	Zamiana tetrad (nieobowiązkowa).
LSL	R0	1	Przesunięcie bitowe rejestru R0 o jeden bit w lewo.
...	...	...	Powtórzenie operacji przesunięcia.
AND/ OR	R2,R0	1	Właściwa operacja logiczna. Wynik na bicie takim jak pierwszy argument.
		Od 5 do 8	

Tabela 4. Przesunięcie bitu z pozycji *r* na *d* (instrukcje przesunięcia w lewo, prawo i zamiany tetrad)

r\d	0	1	2	3	4	5	6	7
0	0	1	2	2	1	2	3	4
1	1	0	1	2	2	1	2	3
2	2	1	0	1	2	2	1	2
3	2	2	1	0	1	2	2	1
4	1	2	2	1	0	1	2	2
5	2	1	2	2	1	0	1	2
6	3	2	1	2	2	1	0	1
7	4	3	2	1	2	2	1	0

Tabela 5. Przesunięcie bitu z pozycji *r* na *d* (instrukcje obrotu rejestru i zamiany tetrad)

r\d	0	1	2	3	4	5	6	7
0	0	1	2	2	1	2	3	2
1	1	0	1	2	2	1	2	3
2	2	1	0	1	2	2	1	2
3	2	2	1	0	1	2	2	1
4	1	2	2	1	0	1	2	2
5	2	1	2	2	1	0	1	2
6	3	2	1	2	2	1	0	1
7	2	3	2	1	2	2	1	0

Tabela 6. Realizacja algorytmu 4.2. w AVR

Rozkaz	Argumenty	Liczba cykli	Znaczenie
LD	R0,k1	2	Pobranie pierwszego argumentu z pamięci (z adresu k1 do rejestru R0).
ANDI	R0,K1	1	Iloczyn z maską bitową K1.
LD	R1,k2	2	Pobranie drugiego argumentu z pamięci (z adresu k2 do rejestru R1).
ANDI	R1,K2	1	Iloczyn z maską bitową K2.
ADD/MUL	R0, R1	1/2	Dodawanie/Mnożenie. Wynik we fladze Z.
		= 7/8	

Tabela 7. Realizacja iloczynu bitowego za pomocą flagi T

Rozkaz	Argumenty	Liczba cykli	Znaczenie
LD	R0,k1	2	Pobranie pierwszego argumentu z pamięci (z adresu k1 do rejestru R0).
LD	R1,k2	2	Pobranie drugiego argumentu z pamięci (z adresu k2 do rejestru R1).
BST	R0,b1	1	Ustawienie flagi bitu tymczasowego na 1.
SBRS	R1,b2	1/2	Sprawdzenie bitu w drugim rejestrze i pominięcie kolejnej instrukcji, jeśli bit jest 1 (przykład błędnej predykcji skoku).
CLT		1	Skasowanie flagi bitu tymczasowego na 1.
BLD	R2,b3	1	Zapis wyniku do rejestru (operacja nie jest konieczna).
		= 8	

Tabela 8. Zapis do pamięci wyniku z bitu rejestru

Rozkaz	Argumenty	Liczba cykli	Znaczenie
ANDI	R0,K1	1	Logiczny iloczyn w celu wyzerowania nieznaczących bitów.
SWAP	R0	1	Zamiana tetrad (nieobowiązkowa).
LSL/LSR	R0	1	Przesunięcie bitowe rejestru R0 o jeden bit w lewo/prawo.
...	...	...	Powtórzenie operacji przesunięcia.
LD	R1,k1	2	Pobranie z pamięci docelowej komórki dla wyniku (z adresu k1 do rejestru R1)
ANDI	R1,K2	1	Logiczny iloczyn w celu wyzerowania bitu do zapisania.
OR	R1,R0	1	Logiczna suma. Wpisanie wyniku operacji.
ST	K1,R1	2	Zapis wyniku do pamięci.
		Od 7 do 10	

Tabela 9. Zapis do pamięci wyniku poprzez flagę T

Rozkaz	Argumenty	Liczba cykli	Znaczenie
IN	R0, SREG	1	Pobranie rejestru statusowego zawierającego flagę Z do rejestru R0.
BST	R0,b1	1	Zapisanie do flagi T wyniku znajdującego się w rejestrze R0 na bicie b1 (jeśli w rezultacie wcześniejszych działań wynik nie znajduje się w T).
LD	R1,k1	2	Pobranie z pamięci docelowej komórki dla wyniku (z adresu k1 do rejestru R1).
BLD	R1,b2	1	Wpisanie bitu T na pozycję b2 w rejestrze docelowym.
ST	K1,R1	2	Zapis wyniku do pamięci
		= 5/6/7	

Tabela 10. Podsumowanie liczby cykli potrzebnych na realizację algorytmów

Metoda	Cykle obliczeń	Cykle zapisu wyniku	Suma
Operacje logiczne + zapis z maską bitową	5 – 8	7 – 10	12 – 18
Operacje logiczne + zapis przez bit T	5 – 8	6	11 – 14
Operacje arytmetyczne + zapis przez bit T	7 – 8	7	14 – 15
Zastosowanie charakterystycznych cech architektury + zapis przez bit T	8	5	13

W zależności od miejsca przechowywania wyniku przedstawiono sposoby jego zapisu do pamięci. W tabeli 8 pokazano zapis wyniku implementacji 5.1. Tabela 9 to natomiast metoda zapisu wyniku pamiętanego jako stan flagi T, która może być użyta także do zapisu bitu z rejestru (drugi wiersz) lub flagi Z, gdy do rejestru pobrany zostanie rejestr statusowy (pierwszy wiersz).

Podsumowanie

Przedstawione w artykule sposoby realizacji operacji bitowych w systemach mikroprocesorowych pokazują, że architektury operujące na wielu bitach jednocześnie wymagają dużego narzutu obliczeniowego na pobranie ze słowa stanu pojedynczego bitu. Czas wykonania właściwej operacji logicznej na dwóch operandach jest pomijalnie mały w zestawieniu z koniecznością wyłuskania bitu oraz jego zapisu w słowie wynikowym.

W tabeli 10 podsumowano liczbę cykli procesora potrzebnych na zrealizowanie algorytmów obliczeń i zapisu wyniku do pamięci. Zastosowanie bitu T przy zapisie wyniku znacznie przyspiesza operację, a zysk zależy od sposobu zwrócenia wyniku z algorytmu.

Opracowując metodę translacji kodu w języku drabinkowym na kod asemblera, należy przeanalizować różne metody realizacji operacji, często charakterystyczne dla konkretnej architektury. Pozwala to w pełni wykorzystać możliwości mikroprocesora, a także znacznie przyspieszyć realizację zadań.

Pomimo istnienia procesorów jednobitowych [6] takie układy nie są popularne, a zastosowanie bardziej złożonych architektur pozwala na wykonanie przez jedno urządzenie wielu funkcji, nie tylko sterowania, ale i komunikacji oraz wizualizacji [7].

Literatura

[1] Kasprzyk J.: *Programowanie sterowników przemysłowych*. Wydawnictwa Naukowo-Techniczne, Warszawa 2006.

[2] Norma IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages.

[3] Atmel AVR 8-bit Instruction Set. Atmel 2014.

[4] Hill G.: *AVR Control Transfer – AVR Looping*. California State University Long Beach 2009.

[5] Pochopień B.: *Arytmetyka systemów cyfrowych*. Wydawnictwo Politechniki Śląskiej, Gliwice 2000.

[6] MC14500B – Industrial Control Unit. Motorola 1995.

[7] Yin Y., Jin W., Liqiang W.: *Research and Implementation of Embedded Soft PLC system*. International Conference on Intelligent Networks and Intelligent Systems 2012.

mgr inż. Łukasz Herb – Politechnika Śląska, Instytut Informatyki
e-mail: lukasz.herb@polsl.pl

artykuł recenzowany