MINUT 2020 (2), s. 136–148
ISSN 2719-3063

Robert BRZESKI[1]

[1]Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice

Prawidłowe tworzenie rozkazów asemblerowych dla Maszyny W cz.2

Streszczenie. Artykuł ten stanowi kontynuację tematyki rozpoczętej w artykule 'Prawidłowe tworzenie rozkazów asemblerowych dla Maszyny W cz.1'. W bieżącym artykule przedstawiony jest kolejny zestaw błędów popełnianych przez studentów. Tym razem są to błędy związane z bardziej dogłębną wiedzą lub takie które mimo wszystko na zajęciach się pojawiają. W tym artykule są one zebrane w jeden zbiór, wraz z krótkim omówieniem problemu i przedstawieniem rozwiązań prawidłowych. Podsumowaniem jest spis opisanych tu błędów popełnianych przez studentów. Artykuł ten może stanowić więc nie tylko uzupełnienia wiedzy dotychczas posiadanej, ale także swego rodzaju listę kontrolną podczas praktycznej implementacji rozkazów asemblerowych dla Maszyny W.

Słowa kluczowe: rozkaz, asembler, Maszyna W.

1. Wprowadzenie do implementacji rozkazów dla Maszyny W

Bieżący artykuł, jako kontynuacja tematyki zawartej w publikacji 'Prawidłowe tworzenie rozkazów asemblerowych dla Maszyny W cz.1' [1], jest przeznaczony dla studentów mających styczność lub chcących poszerzyć swoją wiedzę z zakresu projektowania rozkazów dla Maszyny W. Zagadnienie to poruszane jest między innymi na wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej, w ramach przedmiotu Postawy Informatyki. Pojęcie Maszyny W odnosi się do idei konstrukcji i działania uproszczonego komputera. Zawiera on w sobie podstawowe, najważniejsze elementy maszyny cyfrowej, zgodnej z obecnie powszechnie używaną architekturą von Neumanna. Na prowadzonych zajęciach, w szczególności podczas laboratorium, studenci używają Maszyny W, w postaci programowego symulatora [Rysunek 1].

Na zajęciach z tego przedmiotu zadaniem studenta jest zaimplementowanie przy użyciu mikrosygnałów rozkazu asemblerowego, zgodnie z otrzymaną treścią opisującą co dokładnie dany rozkaz ma realizować. Przed implementacją rozkazu można utworzyć jego projekt w postaci algorytmu, zapisanego w dowolny sposób (np. słownie lub przy użyciu schematu blokowego, lub za pomocą poszczególnych przesyłów pomiędzy rejestrami lub pamięcią). Taki rozkaz wykonywany jest w kilku osobnych cyklach procesora zwanych taktami. W każdym takcie należy umieścić odpowiedni zestaw sygnałów mikrosterujących i zakończyć go znakiem reprezentującym koniec taktu. Dla obecnie używanego na laboratorium symulatora

Autor korespondencyjny: R. Brzeski (Robert.Brzeski@polsl.pl).
Data wpłynięcia: 01.10.2020.

jest to średnik. Implementacja rozkazu na dostępnym symulatorze realizowana jest w oknie, dostępnym po wybraniu z menu *Plik* → *Nowy* → *Rozkaz*, w następującej postaci [1]:

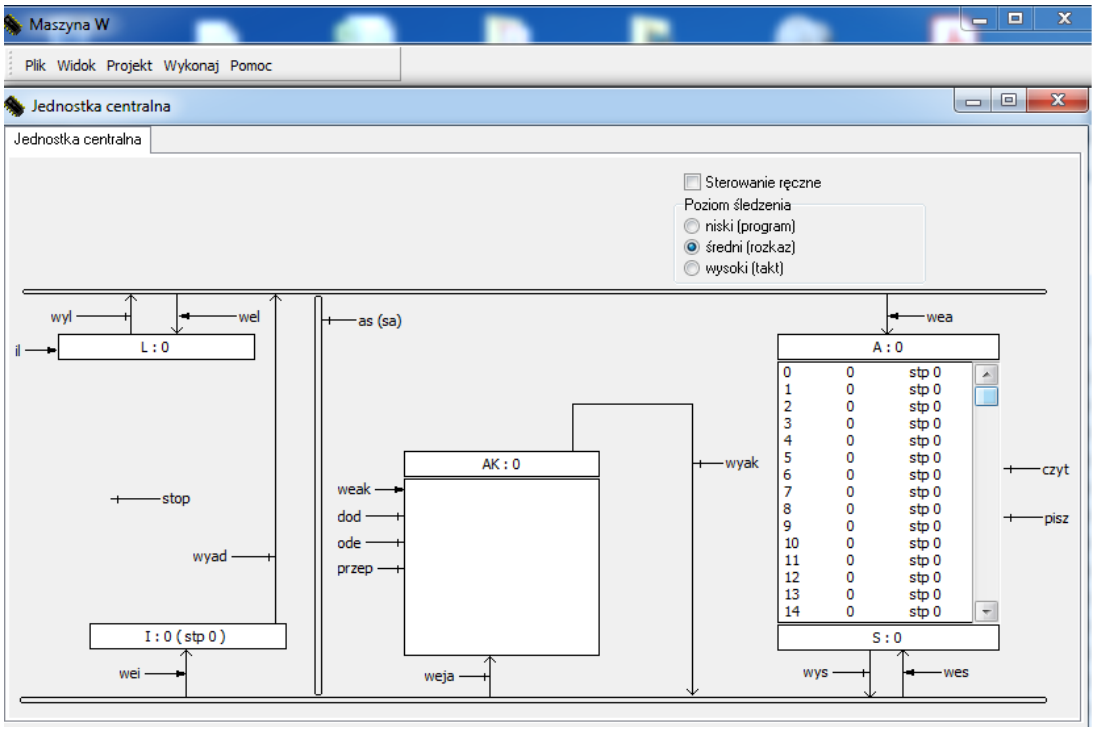
// Dwa znaki ukośnika oznaczają komentarz. Jest on opcjonalny ale warto tu wpisać treść rozkazu która będzie implementowana.

ROZKAZ nazwaRozkazu;
Następnie trzeba umieścić predefiniowane słowo 'ROZKAZ' oraz własną nazwę implementowanego rozkazu. Całość trzeba zakończyć średnikiem.

Argumenty liczbaArgumentów;
Kolejną, tym razem opcjonalną częścią jest wpisanie liczby argumentów obsługiwanych przez implementowany rozkaz. Wartością domyślną jest jeden.

czyt wys wei il;
Pierwszy takt rozkazu. Jest on zawsze taki sam – nie można tu nic dodać ani niczego pominąć. Dla każdego rozkazu składa się z zestawu tych samych czterech mikrosygnałów.

Kolejne takty – zestawy mikrosygnałów;
Każdy takt trzeba zakończyć średnikiem.



Rysunek 1. Widok okna symulatora Maszyny W, w wersji W+

Tak zaimplementowany rozkaz, można skompilować poprzez wybranie opcji *Kompiluj* z menu dostępnego po naciśnięciu prawego przycisku myszy na oknie rozkazu.

Wykorzystanie skompilowanego rozkazu najczęściej odbywa się poprzez użycie go i wykonanie w programie, napisanym w języku asemblera Maszyny W, czyli zestawie tego typu rozkazów [1].

Aby móc użyć, uruchomić napisany rozkaz, musi on być dostępny na liście rozkazów Maszyny W, menu: *Widok* → *Lista rozkazów*. Aby skompilowany rozkaz znalazł się na tej liście, muszą być zmienione domyślne ustawienia, w menu: *Projekt* → *Opcje* → *Liczba bitów kodu* - trzeba zmienić wartość z trzy na cztery. Na trzech bitach można zakodować 8 wartości, czyli tyle ile jest już w Maszynie W rozkazów predefiniowanych. Na czterech bitach można zakodować 16 różnych wartości, które będą reprezentować kody poszczególnych rozkazów - w ten sposób do symulatora Maszyny W, można dodać 8 kolejnych rozkazów.

W trakcie kompilowania rozkazów zostanie zrealizowana wstępna kontrola składniowa zaimplementowanego rozkazu. Jeżeli kompilator znajdzie błąd, to uzyskamy odpowiedni komunikat zwrotny. Jeżeli kompilator będzie w stanie skompilować rozkaz, to zostanie on dodany do listy dostępnych rozkazów: *Widok* → *Lista rozkazów*.

W ramach laboratorium student będzie miał możliwość wdrożenia wiedzy teoretycznej w czasie implementacji zadanych rozkazów asemblerowych. Treść zadań do wykonania będzie przedstawiona w postaci opisu słownego lub w formie skróconej przy użyciu: nazw rejestrów (reprezentowanych przez skróty literowe), nawiasów, wykonywanych operacji lub warunków i wykonywanego przesyłu (przesyłu wartości pomiędzy rejestrami lub pamięcią). Przesył reprezentowany przez znak '→' oznacza, że wartość uzyskana po lewej stronie → jest przesyłana w miejsce wskazywane po prawej stronie →. Skróty literowe oznaczają poszczególne rejestry. W najprostszej sytuacji np.: (A) → B oznacza, że wartość rejestru A należy przesłać do rejestru B [1].

Ważne jest, aby utworzony rozkaz nie tylko działał prawidłowo, czyli realizował postawione przed nim zadanie i był możliwy do użycia w programie, ale także ważne jest, aby był optymalny, czyli był wykonywany w najmniejszej możliwej liczbie taktów (cyklów) procesora.

Podobnie jak w poprzednim artykule [1], tak i w tym zakłada się, że czytelnik posiada podstawową wiedzę teoretyczną na temat Maszyny W [1–6] i poprzez bieżący artykuł chce poszerzyć ją o elementy praktycznych podpowiedzi i dodatkowych wyjaśnień z zakresu implementacji rozkazów. Pamiętać także należy, że jedynie samodzielna realizacja zadań pozwala w pełni zrozumieć zależności pomiędzy elementami tematyki związanej z Maszyną W. Kopiowanie rozwiązań od 'sąsiada' nie skutkuje kopiowaniem umiejętności ani wiedzy, a co najwyżej kopiowaniem błędów tam istniejących. Artykuł ten jest efektem kilkunastoletnich doświadczeń autora w prowadzeniu zajęć z Podstaw Informatyki, a przedstawione w rozdziale 2 nieprawidłowości w trakcie tworzenia rozkazów, są oparte na faktycznie popełnianych przez studentów błędach.

2. Przykłady błędów przy implementacji rozkazów

W bieżącym rozdziale zostaną zaprezentowane przykłady implementacji nieprawidłowych, wraz z ich omówieniem oraz przedstawieniem rozwiązania prawidłowego.

2.1. Błąd związany z brakiem zrozumienia liczby nawiasów wokół rejestru.

Omówienie zagadnienia:

Nawiasy występują wokół skrótów literowych rejestru. Liczba nawiasów ma kluczowe znaczenie. Inaczej jest rozumiana po lewej, a inaczej po prawej stronie przesyłu. Wyjaśnienie liczby nawiasów przedstawione jest na przykładowym rejestrze o nazwie R [1]:

W zależności od liczby nawiasów **po lewej stronie przesyłu**:

(R) – oznacza wartość znajdującą się **w rejestrze R**.

((R)) – oznacza wartość znajdującą się **w pamięci** o adresie wskazywanym przez rejestr R.

(((R))) – oznacza, że wartość rejestru R jest adresem (**wskaźnik**iem) do komórki pamięci, którego wartość ponownie jest **wskaźnik**iem do pamięci, spod którego należy odczytać (pobrać) wartość.

Natomiast **po prawej stronie przesyłu**:

(R) - oznacza, że uzyskaną wcześniej wartość, należy zapisać do pamięci o adresie wskazywanym przez rejestr R (adresie umieszczonym w rejestrze R).

((R)) - oznacza, że wartość rejestru R jest adresem (wskaźnikiem) do komórki pamięci, którego wartość ponownie jest wskaźnikiem do pamięci, pod który należy zapisać wartość 'operacji' uzyskaną z lewej strony przesyłu '→'.

Błędny przykład 1.1 (Wartość rejestru akumulatora dodać do wartości komórki pamięci o adresie znajdującym się w rejestrze AD, wynik dodawania pozostawić w rejestrze akumulatora):

```
// ((AD)) + (AK) → AK
ROZKAZ JJ-J;
czyt wys wei il;
wyad sa weja dod weak; // błąd - AD jest w dwóch nawiasach, a nie w jednym
wyl wea;
```

W tym błędnym przykładzie, niezgodnie z treścią zadania, do akumulatora dodawana jest wartość przechowywana w rejestrze AD. Zgodnie z treścią zadania, symbol rejestru AD jest w dwóch nawiasach, więc w rejestrze AD jest adres komórki pamięci, w której znajduje się wartość, którą to należy dodać do akumulatora. Dlatego w prawidłowym rozwiązaniu, wartość rejestru AD najpierw przesyłana jest do rejestru A (*wyad wea*), w kolejnym takcie jest odczytywana (sygnał *czyt*) wartość komórki pamięci (o adresie umieszczonym w rejestrze A), co powoduje umieszczenie tej wartości w rejestrze S, i następnie przesyłana jest do jednostki arytmetyczno logicznej (*wys weja*) i dodawana do wartości akumulatora (*dod weak*).

Prawidłowe rozwiązanie 1.1:

```
// ((AD)) + (AK) → AK
ROZKAZ JJ-J;
czyt wys wei il;
wyad wea;
czyt wys weja dod weak wyl wea;
```

Gdyby symbol rejestru AD był w trzech nawiasach, to dodatkowo należałoby przesłać odczytaną wartość z pamięci do rejestru A (*wys sa wea*) i w kolejnym takcie ponownie odczytać (*czyt*) wartość z pamięci.

Prawidłowe rozwiązanie 1.2 (Wartość rejestru akumulatora dodać do wartości komórki pamięci o adresie znajdującym się w komórce pamięci o adresie znajdującym się w rejestrze AD, wynik dodawania pozostawić w rejestrze akumulatora):

// **(((AD)))** + (AK) → AK
ROZKAZ JJ-D;
czyt wys wei il;
wyad wea;
czyt **wys sa wea**; // dodatkowy takt realizujący 'trzeci nawias'
czyt wys weja dod weak wyl wea;

2.2. Błąd związany z niezrozumieniem lub zbytnim uproszczeniem treści zadania.

Omówienie zagadnienia: Czasami zdarza się, że student błędnie zrozumie lub nieodpowiednio uprości treść zadania. Zadanie do wykonania trzeba odpowiednio przeanalizować przed jego wykonaniem. Powinno się najpierw znaleźć algorytm realizujący potrzebne czynności i wtedy dopiero przejść do implementacji rozkazu. Przy analizie zadania należy wziąć pod uwagę, że wartości rejestrów które trzeba w zadaniu uwzględnić są z momentu rozpoczęcia wykonywania rozkazu, natomiast dostęp do wartości rejestrów, uzyskuje się dopiero po wykonaniu pierwszego taktu. Ten pierwszy takt jest standardowy dla każdego rozkazu i nie można go zmienić (ani nie można dodać kolejnych mikrosygnałów, ani nie można usunąć któregoś).

Błędny przykład 2.1 (Wartość rejestru AK dodać do wartości rejestru L, wynik dodawania pozostawić w rejestrze akumulatora oraz przesłać do rejestru licznika):

// (L) + (AK) → AK, L
ROZKAZ JD-J;
czyt wys wei **il**; // w tym takcie wartość licznika zostaje zmieniona – zwiększona o 1
wyl sa weja **dod** weak; // błąd - dodawana jest już zmieniona wartość licznika
wyak sa wel wea;

W związku z tym, że wartość licznika już została w pierwszym takcie zmieniona – zwiększona o 1, to w drugim takcie dodawana jest już ta zinkrementowana o 1 wartość licznika, a nie ta która była tam w momencie rozpoczęcia wykonywania rozkazu. Dlatego wynik końcowy jest błędny – jest zwiększony o jeden. W związku z tym, że nie ma bezpośredniego dostępu do wartości początkowej licznika, to uzyskany wynik dodawania, aby był prawidłowy, należy zdekrementować o wartość jeden. Można to zrobić używając możliwości inkrementacji licznika (sygnałem *il*) w ten sposób, że do akumulatora zostanie dodana bieżąca wartość rejestru L (*wyl sa weja dod weak*) a następnie, po inkrementacji wartości licznika, jego wartość zostanie odjęta od akumulatora (*wyl sa weja ode weak*). W ten sposób pierwotna wartość akumulatora zostanie zmniejszona właśnie o wartość 1: (AK) + (L) - [(L)+1] = (AK) – 1.

Prawidłowe rozwiązanie 2.1:

// (L) + (AK) → AK, L
ROZKAZ JD-J;
czyt wys wei il;
wyl sa weja **dod** weak;
wyl sa weja **dod** weak **il**;
wyl sa weja **ode** weak;
wyak sa wel wea;

Warto tu jednak podkreślić i zapamiętać, że w ogólności nie można w dowolny sposób zmieniać zawartości licznika podczas implementacji rozkazu. Jeżeli jest to robione, to najczęściej należy zapamiętać jego wartość pierwotną (po inkrementacji w pierwszym takcie), tak aby można go było odtworzyć. Jednak w tym przypadku nie jest to konieczne, gdyż jest to rozkaz skoku, w którym to w sposób świadomy modyfikujemy wartość licznika, zatem nie ma tu potrzeby zapamiętywania (i odtwarzania) jego pierwotnej wartości.

2.3. Błąd związany z myleniem rejestru AD z rejestrem A.

Omówienie zagadnienia:
Część adresowa rejestru instrukcji reprezentowana symbolem AD często jest mylona z rejestrem A. Odczytać wartość z rejestru AD można sygnałem *wyad*, który udostępnia tę wartość na magistrali adresowej i z niej można przesłać ją dalej, zgodnie z treścią zadania i implementowanym algorytmem.

Błędny przykład 3.1 (Wartość rejestru akumulatora dodać do wartości komórki pamięci o adresie znajdującym się w rejestrze AD, wynik dodawania pozostawić w rejestrze akumulatora):

// ((AD)) + (AK) → AK
ROZKAZ JT-J;
czyt wys wei il;
czyt wys weja dod weak wyl wea;//błąd- do rejestru A najpierw należy przesłać wartość rejestru AD

Prawidłowe rozwiązanie 3.1:

// if (AK) < (AD) then (L)+3 → L else (L)+1 → L
// ((AD)) + (AK) → AK
ROZKAZ JT-J
czyt wys wei il;
wyad wea;
czyt wys weja dod weak wyl wea;

2.4. Błąd związany z nieuprawnionym korzystaniem z pamięci.

Omówienie zagadnienia:
Korzystanie z pamięci, czyli odczytywanie lub zapisywanie wartości pod jakiś adres, może odbywać się tylko wtedy, gdy jest to jawnie określone w treści zadania. Dodatkowo tego typu operacje mogą być

realizowane tylko na tych adresach, które zostały wskazane w treści zadania. Aby nie generować niepotrzebnych błędów w zakresie rozumienia operacji na pamięci, potrzebne jest rozumienie liczby nawiasów wokół rejestru, omówione w rozdziale 2.1. Pamiętać należy, że przy tworzeniu algorytmu dla otrzymanego zadania, nie można samowolnie zapisywać wartości pod jakikolwiek adres w pamięci. Pod każdym adresem w pamięci może znajdować się inny kod rozkazu lub zmienna wykorzystywana programie.

Błędny przykład 4.1 (Od wartości komórki pamięci o adresie znajdującym się w rejestrze AD odjąć wartość rejestru akumulatora, wynik pozostawić w rejestrze akumulatora):

```
// ((AD)) - (AK) → AK
ROZKAZ JC-J;
czyt wys wei il;
wyl wea wyak wes;
pisz; // błąd - nieuprawniony zapis do pamięci
wyad wea;
czyt wys weja przep weak;
wyl wea;
czyt wys weja ode weak;
```

W tym nieprawidłowym przykładzie, wartość akumulatora została zapisana do komórki pamięci, o adresie znajdującym się w liczniku, aby zapamiętać wartość akumulatora, którą student planuje odjąć od wpisanej najpierw do AK (niszcząc pierwotną jego wartość) wartości komórki pamięci o adresie znajdującym się w rejestrze AD. Niestety, zapisując w pamięci (pod adres znajdujący się w liczniku) pierwotną wartość AK, kod programu znajdujący się w tej komórce pamięci (domyślnie następny rozkaz) został nadpisany/zniszczony. W związku z tym nie będzie możliwości wykonania rozkazu następnego po bieżącym, a maszyna W jako następny będzie realizowała rozkaz, będący interpretacją liczby - początkowej wartości akumulatora sprzed wykonania błędnie zrealizowanego rozkazu JC-J.

W prawidłowym rozwiązaniu najpierw uzyskiwana jest ujemna wartość akumulatora, a dopiero wtedy dodawana jest do niego wartość komórki pamięci, o adresie zawartym w rejestrze AD. W takim przypadku nie ma potrzeby zapisywania w pamięci pierwotnej wartości akumulatora.

Prawidłowe rozwiązanie 4.1:

```
// ((AD)) - (AK) → AK
ROZKAZ JC-J;
czyt wys wei il;
wyak wes weja ode weak; // zapisanie wartości AK do rejestru S i jednoczesne zerowanie AK
wys weja ode weak wyad wea; // odjęcie od wyzerowanego akumulatora wcześniejszej jego wartości
czyt wys weja dod weak wyl wea;
```

Błędny przykład 4.2 (Od wartości rejestru AD odjąć jeden. Wynik należy umieścić w rejestrze akumulatora):

Uwaga napisana przez studenta do poniższego błędnego rozkazu: 'pod adresem 1 musi być wartość 1'. Takie założenie jest nieuprawnione, nie można zakładać, że pod jakimś adresem w pamięci będzie przydatna w algorytmie wartość.

// (AD) - 1 → AK
ROZKAZ JC-D;
czyt wys wei il;
wyad sa weja przep weak;
wyl wea;
czyt wys weja ode weak;
wyl wea;

W tym nieprawidłowym rozwiązaniu student nie tylko w sposób nieuprawniony korzysta z pamięci, ale jednocześnie zakłada, że wartość licznika będzie równa jeden, co będzie prawdą jedynie dla pierwszego rozkazu umieszczonego w całym programie.

W prawidłowym rozwiązaniu wynik jest dekrementowany o wartość 1 przy użyciu licznika, podobnie jak w przykładzie 2.1. Dodatkowo wartość licznika jest zachowywana w rejestrze S, gdyż w tym rozkazie nie może ona ulec utraceniu i na końcu trzeba tę wartość przywrócić (*wys sa wel*).

Prawidłowe rozwiązanie 4.2:

// (AD) - 1 → AK
ROZKAZ JC-D;
czyt wys wei il;
wyad sa weja przep weak;
wyl sa weja **dod** weak wea wes **il**;
wyl sa weja **ode** weak;
wys sa wel;

2.5. Błąd związany z nieprawidłowym wyborem architektury Maszyny W.

Omówienie zagadnienia:

Maszyna W, poprzez dodawanie do niej kolejnych komponentów, umożliwia zmianę swojej architektury. Najbardziej podstawowa to architektura W oraz W+ (zawierająca dodatkowo połączenie między magistralą danych i adresową). Zadania do realizacji zazwyczaj ustalane są na konkretną architekturę i samowolnie nie można używać elementów dodatkowych, zawartych w innej architekturze. Oczywiście zadań typowych dla architektury W+ nie da się wykonać na architekturze W. Jeżeli jawnie nie jest podane, na jaką architekturę należy zaimplementować rozkaz, to można powiedzieć, że zadanie powinno się wykonać, w na tyle prostej architekturze na ile to możliwe.

Błędny przykład 5.1: (Wartość rejestru akumulatora dodać do wartości komórki pamięci o adresie znajdującym się w rejestrze AK, wynik dodawania pozostawić w rejestrze akumulatora):

```
// (AK) + ((AK)) → AK          tego zadania nie da się zrealizować na architekturze W
ROZKAZ JP-J;
czyt wys wei il;
wyak wei; // błąd opisany w artykule [1]
wyad wea;
czyt wys weja weak dod wyl wea;
```

Prawidłowe rozwiązanie 5.1:

```
// (AK) + ((AK)) → AK          Do realizacji tego zadania trzeba wykorzystać architekturę W+
ROZKAZ JP-J;
czyt wys wei il;
wyak sa wea; // wartość przesyłana jest przez dodatkowe połączenie międzymagistralowe
czyt wys weja weak dod wyl wea;
```

Błędny przykład 5.2 (Od wartości komórki pamięci o adresie znajdującym się w rejestrze L odjąć wartość rejestru akumulatora, wynik pozostawić w rejestrze akumulatora):

```
// ((L)) - (AK) → AK          Do tego zadania nie trzeba W+, wystarczy W
ROZKAZ JP-D;
czyt wys wei il;
wyl wea;
wyak sa wel;
czyt wys weja przep weak;
wyl sa weja weak ode weak wyl wea;
```

W tym błędnym przykładzie poprzez przesłanie akumulatora do licznika, następuje utrata jego wartości - błąd opisany w artykule [1]. Dodatkowo w ostatnim takcie występuje podwójnie sygnał *wyl* - w obecnie używanym symulatorze Maszyny W nie generuje to żadnego błędu (symulator traktuje to jak pojedyncze wywołanie sygnału *wyl*), jednak kolejne wypisanie w tym samym takcie tego samego sygnału jest nadmiarowe i nie powinno mieć miejsca.

Prawidłowe rozwiązanie 5.2:

```
// ((L)) - (AK) → AK
ROZKAZ JP-D;
czyt wys wei il;
wyl wea wyak weja ode weak wes;
wys weja ode weak;
czyt wys weja dod weak;
```

Dla rozwiązania prawidłowego, dodatkowo warto zwrócić uwagę na brak w ostatnim takcie sygnałów *wyl wea*. W tym rozkazie przesłanie wartości licznika do rejestru adresowego zostało już zrealizowane w drugim takcie.

2.6. Błąd związany z zamianą ścieżek instrukcji 'JEżELI'.

Omówienie zagadnienia:

Warunkowa instrukcja JEżELI zawiera dwie alternatywne ścieżki rozwiązań. Czasami zdarza się pomyłka interpretacji, która ścieżka kiedy jest wykonywana (przy spełnieniu którego warunku). Dotyczy to zarówno sprawdzania sygnału Z (wartość ujemna lub nieujemna) jak i sygnału ZAK (wartość zero lub nie zero). Przy tego typu zadaniach należy upewnić się, czy przy danym warunku, wykonywana jest odpowiednia ścieżka operacji.

Błędny przykład 6.1 (Mniejszą z dwóch wartości należy umieścić w rejestrze akumulatora. Pierwsza z tych dwóch wartości znajduje się w rejestrze AK, natomiast druga wartość w rejestrze AD):

// min{ (AK), (AD) } → AK
ROZKAZ JS-J;
czyt wys wei il;
wyad sa wes weja ode weak
JEżELI z TO @wieksza GDY NIE @mniejsza;
@wieksza wys weja **przep** weak wyl wea KONIEC; // błąd
@mniejsza wys weja **dod** weak wyl wea; // błąd

W tym błędnym przykładzie operacje przepisania (*przep*) oraz dodawania (*dod*) są wykonane przy odwrotnych warunkach, w trakcie innej - tej drugiej ścieżki wykonania instrukcji JEżELI. W wyniku czego zamiast rozkazu wyznaczającego minimum, otrzymamy rozkaz wyznaczający maksimum.

W rozwiązaniu prawidłowym, działanie przepisania jest wykonane w ścieżce oznaczonej etykietą *@mniejsza* zamiast *@wieksza*, natomiast działanie dodawania - odwrotnie.

Prawidłowe rozwiązanie 6.1:

// min{ (AK), (AD) } → AK
ROZKAZ JS-J;
czyt wys wei il;
wyad sa wes weja ode weak
JEżELI z TO @wieksza GDY NIE @mniejsza;
@wieksza wys weja **dod** weak wyl wea KONIEC;
@mniejsza wys weja **przep** weak wyl wea;

Błędny przykład 6.2 (Jeżeli wartość znajdująca się w akumulatorze jest równa zero, wtedy początkową wartość licznika należy zwiększyć o wartość trzy, w przeciwnej sytuacji wartość licznika należy zwiększyć o wartość jeden):

// if (AK) = 0 then (L) + 3 → L else (L) + 1 → L
ROZKAZ JS-D;
czyt wys wei il;
JEżELI zak TO @**jeden** GDY NIE @**trzy**; // błąd
@trzy il;
il DALEJ @jeden;

@jeden wyl wea;

W tym błędnym przykładzie zamieniony został warunek wykonania poszczególnych ścieżek instrukcji JEżELI.

W prawidłowym rozwiązaniu etykiety poszczególnych ścieżek przy instrukcji JEżELI są umieszczone odwrotnie, co spowoduje poprawne wykonanie tego rozkazu: gdy wartość znajdująca się w akumulatorze jest równa zero, należy pominąć realizację dwóch następnych rozkazów zapisanych w programie, a w przeciwnej sytuacji przejść do realizacji kolejnego rozkazu programu.

Prawidłowe rozwiązanie 6.2:

// if (AK) = 0 then (L) + 3 → L else (L) + 1 → L
ROZKAZ JS-D;
czyt wys wei il;
JEżELI zak TO @**trzy** GDY NIE @**jeden**;
@trzy il;
il DALEJ @jeden;
@jeden wyl wea;

2.7. Błąd związany z próbą umieszczania w liczniku wartości ujemnej.

Omówienie zagadnienia:

W związku ze sposobem zapisu liczb ujemnych w trybie dopełnieniowym do 2, stosowanym typowo w maszynach cyfrowych, w liczniku nie można przechowywać wartości ujemnych. W Maszynie W przesłanie wartości ujemnej do licznika, powoduje uzyskanie w liczniku innej wartości dodatniej. Zdecydowanie nie należy wykonywać takiej operacji.

Błędny przykład 7.1 (Podwójną wartość rejestru AD zdekrementować o jeden i taki wynik umieścić w liczniku. Dodatkowe założenie: wartość akumulatora nie musi być zachowana):

// 2*(AD) - 1 → L
ROZKAZ JSI-J;
czyt wys wei il;
wyl sa weja przep weak il;
wyl sa weja ode weak;
wyak sa wel; // błąd - w tym momencie nastąpi zmiana wartości z minus 1 na wartość
// maksymalną dodatnią możliwą do przechowania w liczniku
wyad sa weja przep weak;
wyak weja dod weak;
wyl was weja ode weak;
wyak sa wel wea;

W błędnym przykładzie niepotrzebnie utworzono wartość minus jeden. Wartość ta w akumulatorze kodowana jest standardowo na 8 bitach, natomiast po przesłaniu jej do licznika (poprzez magistralę adresową, która jest węższa od magistrali danych) otrzymujemy standardowo wartość 5-bitową z obciętymi

trzema najbardziej znaczącymi bitami. Zagadnienie problemu przesyłów między magistralami Maszyny W i utrata bitów jest dokładniej opisana w [5] (str.6-7).

W prawidłowym rozwiązaniu najpierw uzyskano podwojoną wartość rejestru AD, a dopiero w następnym kroku zmniejszono tę wartość o jeden. Zmniejszenie wartości o jeden zrealizowano analogicznie jak w przykładzie 4.2

Prawidłowe rozwiązanie 7.1:

// 2*(AD) - 1 → L
ROZKAZ JSI-J;
czyt wys wei il;
wyad sa weja przep weak;
wyak weja dod weak;
wyl sa weja dod weak il;
wyl sa weja ode weak;
wyak sa wel wea;

3. Wnioski końcowe

Realizując zadanie utworzenia rozkazu, czyli znalezienia odpowiedniego algorytmu i jego implementacji, można popełnić wiele różnych błędów. Wynikają one najczęściej z braku jakiegoś elementu wiedzy, ale czasami także są skutkiem zastosowania nieodpowiedniego algorytmu lub zwykłej pomyłki. Aby ułatwić, zarówno unikanie tych błędów jak i znalezienie prawidłowych rozwiązań, można utworzyć listę błędów mających miejsce podczas tworzenia rozkazu:

1. Brak zrozumienia liczby nawiasów wokół rejestru - należy w pełni rozumieć zapis treści rozkazu w postaci nawiasowej.

2. Niezrozumienie lub zbytnie uproszczenie treści zadania - należy dokładnie przeczytać i przeanalizować treść zadania.

3. Mylenie rejestru AD z rejestrem A - są to zupełnie inne, osobne rejestry.

4. Nieuprawnione korzystanie z pamięci (zapis, odczyt) - jeżeli w treści zadania nie ma takiego polecenia, to nie można korzystać z pamięci.

5. Nieprawidłowy wybór architektury Maszyny W - zadania należy wykonać na takiej architekturze na jaką zostało to zlecone. Jeżeli nie ma jawnej informacji o wyborze architektury, to należy zaimplementować rozkaz w na tyle prostej architekturze na ile to możliwe.

6. Zamiana ścieżek instrukcji 'JEżELI' - należy upewnić się, czy przy danym warunku, wykonywana jest odpowiednia ścieżka operacji.

7. Błąd związany z przesyłaniem do licznika wartości ujemnej - w liczniku nie można umieszczać wartości ujemnych.

Artykuł ten, w szczególności dla studentów, może być nie tylko materiałem pomocnym przy realizacji otrzymanych zadań i rozwiązywaniu potencjalnych problemów, ale także, przy odpowiednim opanowaniu zawartych tu wskazówek, pozwala tych problemów uniknąć. Stanowi także dobre narzędzie dla studentów do sprawdzenia stopnia opanowania umiejętności praktycznych i prawidłowej implementacji rozkazów.

Podziękowania

Autor pragnie podziękować recenzentom za trud włożony w recenzje.

Literatura

1. R. Brzeski, *Prawidłowe tworzenie rozkazów asemblerowych dla Maszyny W cz.1*, MINUT 2020 (2), s. 124-135.
2. M. Chłopek, R. Tutajewicz, *Wykłady z podstaw Informatyki profesora Stefana Węgrzyna*, Skrypt uczelniany Politechniki Śląskiej nr 2062, Wydawnictwo Politechniki Śląskiej, Gliwice, 1997.
3. K. Grochla, G. Hryń, S. Iwaszenko, P. Kasprzyk, J. Kubica, M. Widera, T. Wróbel, *Wykłady z podstaw Informatyki profesora Stefana Węgrzyna*, Skrypt uczelniany Politechniki Śląskiej nr 2321, Wydawnictwo Politechniki Śląskiej, Gliwice, 2003.
4. A. Momot, R. Tutajewicz, *Maszyna W - jak zaprojektować prosty rozkaz*, MINUT 2019 (1), s. 24-35.
5. A. Momot, *Projektowanie rozkazów dla maszyny W - konspekt ćwiczeń laboratoryjnych*, MINUT 2020 (2), s. 1-11.
6. S. Węgrzyn, *Podstawy informatyki*, PWN, Warszawa, 1982.