

Robert BRZESKI<sup>1</sup>

<sup>1</sup>Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice

## Prawidłowe tworzenie rozkazów assemblerowych dla Maszyny W cz.1

**Streszczenie.** W artykule przedstawiono przykłady implementacji kilku rozkazów assemblerowych dla Maszyny W wraz z omówieniem błędów realizowanych przez studentów. Pokazano typowe problemy występujące przy implementacji rozkazów oraz zaprezentowano i omówiono prawidłowe rozwiązania. W zakresie prezentacji błędów popełnianych przez studentów skupiono się na tych najczęściej występujących. Przedstawiono także w punktach, zakres materiału teoretycznego potrzebnego do opanowania, jeszcze przed przystąpieniem do realizacji zadań praktycznych. Zestaw ten może być przydatny jako lista kontrolna wstępnie wymaganej wiedzy.

**Słowa kluczowe:** rozkaz, assembler, optymalizacja, Maszyna W.

### 1. Wprowadzenie do implementacji rozkazów dla Maszyny W

W procesie dydaktycznym realizowanym na wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej, na kilku kierunkach, w ramach przedmiotu Postawy Informatyki, realizowane jest zagadnienie projektowania rozkazów dla Maszyny W. 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]. Zadaniem studentów jest zaimplementowanie przy użyciu mikrosygnatów, pojedynczego rozkazu assemblerowego. 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 jest to średnik. Implementacja rozkazu na dostępnym symulatorze realizowana jest w następującej postaci:

```
// 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 mikro sygnałów.

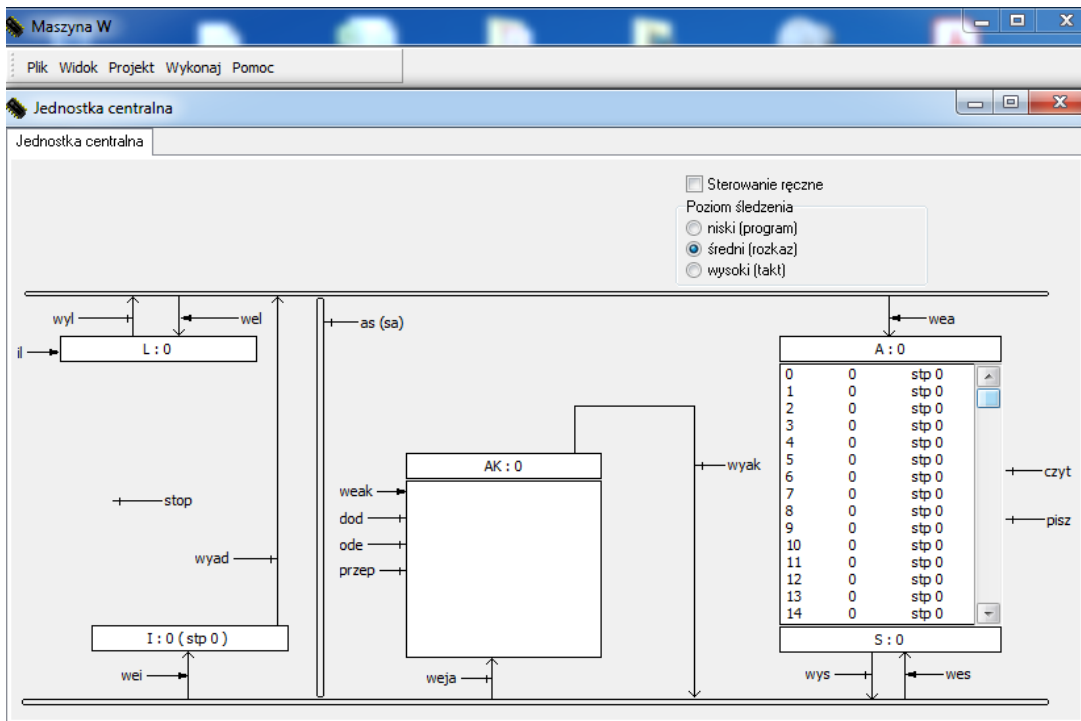
**Kolejne takty – zestawy mikro sygnałów;**

Każdy takt trzeba zakończyć średnikiem.

Tak zaimplementowany rozkaz, można skompilować na dostępnym symulatorze i uruchomić/wykonać. Wykorzystanie skompilowanego rozkazu najczęściej odbywa się poprzez użycie go i wykonanie w programie, napisanym w języku asemblera Maszyny W, czyli zestawie tego typu rozkazów.

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).

Celem tego artykułu nie jest przedstawianie podstawowych informacji na temat struktury i działania Maszyny W. Taką podstawową wiedzę można uzyskać nie tylko na wykładach, ćwiczeniach czy poprzez zaznajomienie się z materiałami do laboratorium z tego tematu, ale także w opublikowanych artykułach i skryptach [1–5]. W bieżącym artykule zakłada się, że czytelnik ma już podstawową wiedzę teoretyczną i chciałby poszerzyć ją o praktyczne wskazówki dotyczące implementacji rozkazów.



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

W ramach tego artykułu przyjmuje się, że czytelnik ma taką wiedzę jaką powinien mieć student przystępujący do laboratorium z opisywanej tematyki, czyli:

1. Zna rejestry wchodzące w skład Maszyny W: S, AK, I, AD, L, A.
2. Zna mikro sygnały sterujące tymi rejestrami.
3. Rozumie różnicę pomiędzy sygnałem poziomym a impulsowym.
4. Zna operacje jakie mogą być wykonywane w JAL (Jednostce Arytmetyczno Logicznej).
5. Umie obsługiwać dostępną pamięć (zapisywanie, odczytywanie wartości).
6. Wie gdzie znajduje się magistrala danych, adresowa oraz połączenie między magistralami, wraz z mikro sygnałem sa.
7. Rozumie ideę i konstrukcję skoku warunkowego 'JEZELI', opartego na testowaniu bitu Z lub ZAK.
8. Zna strukturę rozkazu oraz ideę działania podstawowych, predefiniowanych rozkazów dla Maszyny W (np. rozkazu DOD, SOM, SOB).
9. Rozumie wartości binarne ze znakiem w zapisie uzupełnieniowym do 2.
10. Rozumie ideę podziału/grupowania mikro sygnałów na takty oraz to dlaczego nie jest możliwe wykonanie całego rozkazu w jednym takcie.
11. Zna strukturę wewnętrzną rejestru instrukcji (I, część kodu rozkazu, część adresowa AD).
12. Rozumie sposób zapisu treści rozkazu, w tym między innymi interpretację liczby nawiasów w których umieszczone są nazwy rejestrów.

Powyższe założenia mogą stanowić dla studenta, swego rodzaju listę kontrolną, z zakresu wiadomości potrzebnych do opanowania, jeszcze przed przystąpieniem do ćwiczeń laboratoryjnych. W ramach laboratorium student będzie miał możliwość wdrożenia wiedzy teoretycznej w czasie implementacji zadanych rozkazów assemblerowych. 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. 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:

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 (**wskaznikiem**) do komórki pamięci, którego wartość ponownie jest **wskaznikiem** 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 (wskaznikiem) do komórki pamięci, którego wartość ponownie jest wskaznikiem do pamięci, pod który należy zapisać wartość ‘operacji’ uzyskaną z lewej strony przesyłu ‘→’.

Artykuł ten jest efektem kilkunastoletnich doświadczeń autora w prowadzeniu zajęć z Podstaw Informatyki, a przedstawione w rozdziale 2 i 3 nieprawidłowości w trakcie tworzenia rozkazów, są oparte na faktycznie popełnianych przez studentów błędach.

## 2. Ogólne wytyczne dotyczące prawidłowej implementacji rozkazów

W trakcie realizacji przez studentów otrzymanych zadań, dość często pojawia się kilka ogólnych zagadnień utrudniających implementację rozkazu.

Jednym z problemów jakie napotykają studenci jest przeświadczenie o braku wystarczającej liczby dostępnych rejestrów. Wynika to np. z próby zachowania wartości pierwotnych - występujących w rejestrach w momencie rozpoczęcia wykonywania pierwszego taktu rozkazu. Zanim podejmie się próbę zachowania wartości pierwotnej, należy przeanalizować treść zadania do wykonania (czynności wykonywane w trakcie realizacji rozkazu) i wywnioskować czy taka potrzeba w ogóle istnieje. Być może dana wartość i tak będzie zmieniona i nie ma potrzeby, aby ją przechowywać. Jeżeli nie wynika to jawnie z treści zadania, to wartości pierwotnych rejestru AK, A oraz S nie trzeba zachowywać.

Drugi aspekt przeświadczenia o braku wystarczającej liczby dostępnych rejestrów, jest związany z tym, że czasami rzeczywiście nie ma już miejsca (dostępnych rejestrów) do bezpośredniego zachowania wartości, która na końcu rozkazu musi być taka jak wcześniej. W takiej sytuacji należy zastanowić się, czy zamiast zachowywania danej wartości, nie dałoby się jej odtworzyć, np. poprzez wykonanie operacji odwrotnych/przeciwnych, do tych które zmieniły tę potrzebną wartość (jeżeli np. od akumulatora odjęto wartość rejestru S, to, aby przywrócić wartość AK należy do niego dodać wartość rejestru S – oczywiście dotyczy to tylko sytuacji, w której wartość rejestru S nie uległa zmianie).

Kolejny powód przeświadczenia studentów o braku wystarczającej liczbie dostępnych rejestrów i braku możliwości wykonania zadania, wynika z próby implementacji nieprawidłowego lub nieodpowiedniego (nieoptymalnego) algorytmu. W takiej sytuacji należy znaleźć inny sposób rozwiązania. Należy podkreślić, że wszystkie zadanie dawane do realizacji studentom są przetestowane i możliwe do realizacji na dostępnej liczbie rejestrów.

Niektóre rozwiązania studentów są tak abstrakcyjne, nie tylko w tak wielu elementach nieoptymalne ale posiadające całe zestawy błędów, że aż trudno je komentować. Czasami tak bardzo nie wiadomo ‘co student miał na myśli’ (czyli np. w miarę realizacji rozkazu, student wielokrotnie zmieniał koncepcję rozwiązania, lub też raczej nieskutecznie próbował ją znaleźć), tak bardzo brakuje całościowej, spójnej

koncepcji rozwiązania i zrozumienia poszczególnych jego elementów, że jedyną słuszną drogą jest odsłanie studenta do ponownego (a może pierwszego) zaznajomienia się z podstawowymi informacjami dotyczącymi projektowania rozkazu i tematyki Maszyny W.

Czasami zdarzają się błędy drobne typu nieprawidłowa nazwa mikro sygnału, brak średnika w miejscu w którym chciano zakończyć takt, czy też błąd składniowy w instrukcji skoku warunkowego.

Innej kategorii błędem, ale czasami występującym, jest sytuacja w której student zamiast samodzielnej realizacji zadania, kopiuje je z innego źródła. Wynikiem może być pojawienie się u różnych studentów tych samych, często bardzo nietypowych błędnych rozwiązań. W ten sposób student nie daje sobie, ani możliwości zrozumienia występujących uwarunkowań, ani nabycia praktycznych umiejętności w zakresie nie tylko implementacji ale i tworzenia prawidłowo działających, optymalnych algorytmów. Rozwiązaniem w takiej sytuacji może być przekazanie studentowi do realizacji nietypowego lub nowego rozkazu.

Jeżeli student opanował podstawy teoretyczne dotyczące tematu projektowania rozkazów, to czasami i tak pojawiają się błędy pojedyncze. Są one często na tyle typowe, na tyle często powtarzające się, że można zebrać je w zbiór błędów podstawowych i w ten sposób wskazać nie tylko te elementy na które warto zwrócić uwagę, ale dać informację na temat prawidłowego sposobu rozwiązania/implementacji. Tego typu błędy wraz z rozwiązaniami zostały przedstawione szczegółowo w rozdziale 3.

Ogólnie zagadnienie optymalizacji rozkazów, prawidłowego ich tworzenia, polega na tym, aby zaimplementować prawidłowo działający rozkaz, w na tyle na ile to możliwe, najmniejszej liczbie taktów. Czyli parafrazując Alberta Einsteina, należy utworzyć rozkaz na tyle prosty na ile tylko jest to możliwe, ale nie prościej.

### 3. Przykłady błędów przy implementacji rozkazów

W tym rozdziale zostaną zaprezentowane konkretne przykłady, wraz z omówieniem typowych implementacji nieprawidłowych oraz przedstawieniem rozwiązania prawidłowego.

#### 3.1. Błąd związany z brakiem prawidłowego zakończenia rozkazu.

##### Omówienie zagadnienia:

Rozkaz poza realizacją zadania właściwego, musi także przygotować komputer do realizacji kolejnego rozkazu, który przy realizacji całego programu, będzie realizowany jako rozkaz następny. Aby to zrealizować, w rejestrze A musi znaleźć się adres właśnie tego kolejnego rozkazu. Jeżeli rozkaz zgodnie ze swoją treścią, nie wykonuje skoku do wyznaczonego miejsca w programie, to następny rozkaz do wykonania jest kolejnym i wtedy wystarczy, aby w rejestrze A znalazła się wartość z rejestru licznika, który w pierwszym takcie został zwiększony o jeden (mikrosygnałem *il*), czyli właśnie na adres następnego rozkazu po bieżącym. Zwykle realizowane jest to mikro sygnałami *wyl wea*.

**Błędny przykład 1.1** (Wartość rejestru akumulatora dodać do wartości rejestru AD, wynik dodawania pozostawić w rejestrze akumulatora):

```
// (AK) + (AD) → AK
ROZKAZ ZJ-J;
czyt wys wei il;
wyad sa weja dod weak; // błąd - brak zakończenia rozkazu
```

**Prawidłowe rozwiązanie 1.1:**

```
// (AK) + (AD) → AK
ROZKAZ ZJ-J;
czyt wys wei il;
wyad sa weja dod weak;
wyl wea; // w rejestrze A musi znaleźć się wartość rejestru licznika
```

Jeżeli w rozkazie występuje instrukcja skoku warunkowego ('JEŻELI...') to odpowiednie zakończenie rozkazu musi znaleźć się w każdej ścieżce (instrukcji JEŻELI).

**Błędny przykład 1.2** (Sprawdzić która z dwu wartości jest mniejsza i umieścić tę wartość w rejestrze akumulatora. Początkowo pierwsza wartość jest w akumulatorze a druga w rejestrze AD):

```
// min{(AK), (AD)} → AK
ROZKAZ ZJ-D;
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; // w tej ścieżce jest prawidłowe zakończenie
@mniejsza wys weja przep weak; // błąd - brak zakończenia w tej ścieżce
```

**Prawidłowe rozwiązanie 1.2:**

```
// min{(AK), (AD)} → AK
ROZKAZ ZJ-D;
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; // obie ścieżki muszą odpowiednio kończyć rozkaz
```

Jeżeli jednak w bieżącym rozkazie jest do wykonania skok, to wtedy adres spod którego będzie czytany następny rozkaz, musi zostać umieszczony nie tylko w rejestrze A, ale także w rejestrze L, aby można kontynuować program od właśnie tego adresu. Przykładowo, jeżeli odpowiedni adres (zgodnie z treścią rozkazu do implementacji) miałby zostać pobrany z rejestru AD, to należałoby wykonać mikroinstrukcje *wyad wel wea*.

**Błędny przykład 1.3** (Wartość rejestru akumulatora dodać do wartości rejestru AD, a następnie wynik umieścić w rejestrze licznika – w ten sposób następuje skok w inne miejsce pamięci operacyjnej, czyli w inne miejsce umieszczonego w pamięci programu):

```
// (AK) + (AD) → L
ROZKAZ ZJ-T;
czyt wys wei il;
```

wyad sa weja dod weak;  
 wyak sa wel; // błąd - brak zakończenia rozkazu – wartość licznika nie została przesłana do A

### Prawidłowe rozwiązanie 1.3:

```
// (AK) + (AD) → L
ROZKAZ ZJ-T;
czyt wys wei il;
wyad sa weja dod weak;
wyak sa wel wea; // ta sama wartość musi się znaleźć w liczniku i rejestrze A
```

## 3.2. Błąd związany z utratą wartości licznika w trakcie wykonywania rozkazu.

**Omówienie zagadnienia:** Zawartość rejestru licznika (L), przede wszystkim z powodu kontekstu wykonywania całego programu, jest na tyle kluczowa, że zazwyczaj nie można jej utracić. Jedynie w sytuacji, gdy w ramach rozkazu wykonywany jest skok, wartość licznika jest odpowiednio nadpisywana. W innych sytuacjach jest odpowiednio inkrementowana, najczęściej o wartość 1, w porównaniu do wartości jaka była tam w momencie rozpoczęcia wykonywania danego rozkazu (jeszcze przed wykonaniem pierwszego taktu). Nie można więc jej całkowicie utracić, a jeżeli w trakcie wykonywania rozkazu zostanie chwilowo zmieniona, to należy odpowiednią wartość przywrócić.

**Błędny przykład 2.1** (Wartość rejestru AD zdekrementować o jeden i taki wynik umieścić w akumulatorze):

```
// (AD) -1 → AK
ROZKAZ ZD-J;
czyt wys wei il;
wyad sa weja przep weak;
wyl sa weja dod weak wea il; // błąd - utrata wartości licznika – nie został on nigdzie zapisany
wyl sa weja ode weak;
```

### Prawidłowe rozwiązanie 2.1:

Dla tego rozkazu, przed zmianą wartości - zinkrementowaniem licznika o 1, jego wartość pierwotna zapisywana jest w rejestrze S [takt 3 - *wyl sa wes*] i jest przywracana [takt 5 - *wys sa wel*] po wykonaniu innych operacji [takt 4]. W ten sposób ostatecznie wartość licznika nie ulega utracie, mimo wykonania zmiany jego wartości. Dla tego rozkazu warto dodatkowo zauważyć, że w ostatnim takcie wartość licznika nie jest wpisywana do rejestru A. Zostało to już zrealizowane w trzecim takcie mikro sygnałem *wea*. Oczywiście ten sygnał *wea* zamiast w trzecim takcie, mógłby się znaleźć w ostatnim, ważne jest, aby po wykonaniu całego rozkazu, w rejestrze A znajdowała się ta sama wartość co w rejestrze L.

```
// (AD) - 1 → AK
ROZKAZ ZD-J;
czyt wys wei il;
wyad sa weja przep weak;
wyl sa weja dod weak wea wes il; // takt 3
```

```
wyl sa weja ode weak; // takt 4
wys sa wel; //takt 5
```

### 3.3. Błąd związany z nieodpowiednim zakończeniem pierwszej ścieżki instrukcji warunkowej ‘JEŻELI’.

#### Omówienie zagadnienia:

Pierwsza ścieżka instrukcji ‘JEŻELI’ musi być odpowiednio zakończona. Jeżeli rozkaz ma się zakończyć po wykonaniu pierwszej ścieżki, to należy zakończyć go słowem kluczowym **KONIEC** (tak jak w rozdziale 3.1. dla rozkazu ZJ-D [przykład 1.2]). Jeżeli natomiast chcielibyśmy, aby wykonywanie rozkazu było kontynuowane, to należy umieścić słowo kluczowe **DALEJ** wraz z **nazwą etykiety** dla drugiej ścieżki. Bez odpowiedniego słowa kluczowego wystąpi błąd przy kompilacji rozkazu i student czasami traci dużo czasu na znalezienie przyczyny występowania tego błędu.

**Błędny przykład 3.1** (Jeżeli wartość w akumulatorze jest mniejsza od wartości rejestru AD, wtedy początkową wartość licznika należy zwiększyć o wartość 3, w przeciwnej sytuacji początkową wartość licznika należy zwiększyć o jeden):

```
// if (AK) < (AD) then (L)+3 → L else (L)+1 → L
ROZKAZ ZT-J;
czyt wys wei il;
wyad sa weja ode weak
JEŻELI z TO @wieksza GDY NIE @mniejsza;
@wieksza il;
il; // błąd - brak odpowiedniego słowa kluczowego
@mniejsza wyl wea;
```

#### Prawidłowe rozwiązanie 3.1:

```
// if (AK) < (AD) then (L)+3 → L else (L)+1 → L
ROZKAZ ZT-J;
czyt wys wei il;
wyad weja ode weak
JEŻELI z TO @wieksza GDY NIE @mniejsza;
@wieksza il;
il DALEJ @mniejsza; // dodano słowo kluczowe DALEJ oraz nazwę etykiety @mniejsza
@mniejsza wyl wea;
```

### 3.4. Błąd związany z ponownym wpisaniem wartości do rejestru I.

#### Omówienie zagadnienia:

Rejestr I ma specjalne przeznaczenie i w trakcie wykonywania rozkazu, poza pierwszym taktom, nie można do niego wpisywać jakichkolwiek wartości. Można jedynie odczytywać jego część AD. Jeżeli taki błąd student zrobi w trakcie laboratorium, to otrzyma błąd już w trakcie kompilacji rozkazu. Jeżeli natomiast ten błąd pojawi się w czasie egzaminu, to takie zadanie zazwyczaj oceniane jest na ocenę negatywną, bez względu na pozostałą jego zawartość.



**Błędny przykład 4.1** (Do wartości akumulatora należy dodać wartość komórki pamięci, o adresie wskazywanym przez początkową wartość akumulatora. Wynik tego dodawania należy umieścić w rejestrze akumulatora):

```
// (AK) + ((AK)) → AK
ROZKAZ ZC-J;
czyt wys wei il; //pierwszy takt rozkazu
wyak wei; // błąd - poza pierwszym taktom nie można używać mikro sygnału wei
wyad wea;
czyt wys weja weak dod wyl wea;
```

**Prawidłowe rozwiązanie 4.1:**

```
// (AK) + ((AK)) → AK
ROZKAZ ZC-J;
czyt wys wei il;
wyak sa wea; // wartość przesłana jest przez połączenie między magistralą danych a adresową
czyt wys weja weak dod wyl wea;
```

### 3.5. Błąd związany z umieszczaniem mikro sygnałów w nieodpowiednim takcie.

**Omówienie zagadnienia:**

Umieszczanie mikro sygnału w nieodpowiednim takcie może spowodować błędne działanie rozkazu, często zupełnie inne niż student przewidywał. W danym takcie ciąg mikro sygnałów powinien stanowić zamknięty ciąg czynności. Najczęściej jest to przesłanie danej na magistralę i wpisanie tej danej z magistrali do innego rejestru. Magistrala jest jedynie medium transmisyjnym i nie przechowuje wartości umieszczonych tam danych pomiędzy taktami procesora. Nie można więc odczytywać w kolejnym takcie, wartości umieszczonej na magistrali w poprzednim takcie. Nie można także w jednym takcie umieszczać na tej samej magistrali danych z więcej niż jednego rejestru.

**Błędny przykład 5.1:** (Podwójną początkową wartość akumulatora należy dodać do wartości rejestru AD. Wynik tego dodawania należy umieścić w rejestrze akumulatora):

```
// 2*(AK) + (AD) → AK
ROZKAZ ZP-J;
czyt wys wei il;
wyak weja dod weak wyad sa wes; // błąd - jednoczesne używanie tej samej magistrali
wys weja dod weak wyl wea;
```

**Błędny przykład 5.2** (treść rozkazu jak w przykładzie 5.1):

```
// 2*(AK) + (AD) → AK
ROZKAZ ZP-D;
czyt wys wei il;
wyak weja dod weak;
wyad sa; // przesłanie wartości na magistralę danych bez wpisania jej do jakiegokolwiek rejestru
```

**weja** dod weak wyl wea; // błąd - próba odczytu z pustej magistrali

### Prawidłowe rozwiązanie 5.1 i 5.2:

```
// 2*(AK) + (AD) → AK
ROZKAZ ZP-J; // oraz ZP-D
czyt wys wei il;
wyak weja dod weak wyl wea; // na każdą z magistral wpisano tylko jedną wartość
wyad sa weja dod weak; // po wysłaniu wartości na magistralę, w tym samy takcie jest pobierana
```

Warto przy tym podkreślić, że w przedostatnim takcie rozkazu dopisano sygnały *wyl wea*, które powodują poprawne zakończenie tego rozkazu (a przy okazji została zoptymalizowana liczba taktów tego rozkazu). Zwykle tę parę sygnałów dopisujemy do ostatniego taktu. Jednak w tym przypadku ze względu na fakt, że w ostatnim takcie używane są obie magistrale, nie jest to możliwe. Trzeba więc byłoby dopisać dodatkowy takt wraz tymi sygnałami (*wyl wea*), wydłużając tym samym liczbę taktów. Możemy jednak zauważyć, że w przedostatnim takcie magistrala adresowa nie była używana, więc tę parę sygnałów (przygotowującą rejestr adresowy do poprawnego rozpoczęcia kolejnego rozkazu) można było dopisać wcześniej.

**Błędny przykład 5.3** (Jeżeli wartość akumulatorze jest mniejsza od wartości rejestru AD wtedy początkową wartość licznika należy zwiększyć o wartość 3 w przeciwnej sytuacji początkową wartość licznika należy zwiększyć o jeden):

```
// if (AK) < (AD) then (L)+3 → L else (L)+1 → L
ROZKAZ ZP-T;
czyt wys wei il;
wyad weja ode weak
JEŻELI z TO @wieksza GDY NIE @mniejsza;
@wieksza il il DALEJ @mniejsza; // błąd - nie można w tym samym takcie inkrementować
// licznika więcej niż jeden raz
@mniejsza il wyl wea; // błąd opisany poniżej
```

W ostatnim takcie, inkrementacja wartości licznika (*il*) zostanie zrealizowana dopiero po wysłaniu wartości nieinkrementowanej (do A poprzez *wyl wea*). W tym konkretnym przykładzie, student pokazuje jak wielu elementów jeszcze nie rozumie. Zwiększenie licznika w ten sposób najprawdopodobniej spowoduje błędne wyznaczenie kolejnego rozkazu do wykonania, w czasie realizacji następnego rozkazu po bieżącym. Jednocześnie takie użycie inkrementacji licznika, w zależności od treści zadania, może nie realizować wymaganej w bieżącym rozkazie czynności.

### Prawidłowe rozwiązanie 5.3:

```
// if (AK) < (AD) then (L) +3 → L else (L) +1 → L
ROZKAZ ZP-T;
czyt wys wei il;
wyad weja ode weak
```

```

JEZELI z TO @wieksza GDY NIE @mniejsza;
@wieksza il; // po wykonaniu inkrementacji następuje zakończenie taktu
il DALEJ @mniejsza; // kolejna inkrementacja L jest wykonywana niezależnie od poprzedniej
@mniejsza wyl wea; // w tym taktie przesłana do A wartość licznika nie zostaje już zmieniona

```

### 3.6. Błąd związany z nieprawdziwym założeniem co do wartości danego rejestru.

#### Omówienie zagadnienia:

Nie można robić wstępnych (nieprawdziwych) założeń, co do wartości danego rejestru np., że zawartość rejestru licznika jest równa jeden. Niestety studenci często robią takie nieprawidłowe założenia. W kontekście licznika, prawdopodobnie związane jest to z brakiem świadomości szerszego kontekstu wykorzystania tworzonego rozkazu. Rozkazy assemblerowe są tworzone po to, aby tworzyć z nich całe programy. Pojedynczy rozkaz, jeżeli zostanie umieszczony w programie jako pierwszy, to w czasie wykonania jego pierwszego taktu, inkrementuje on licznik do wartości 1. Wynika to z tego, że adresacja pamięci rozpoczyna się od wartości 0 i to od niej Maszyna W zaczyna wykonywać cały program. Kolejne rozkazy będą ponownie zmieniać wartość licznika, zazwyczaj na wartość inną niż 1. Dlatego jedynie rozkaz umieszczony w programie jako pierwszy, uzyska wartość w liczniku równą 1 i tylko wtedy rozkaz zakładający taką wartość w liczniku, mógłby działać prawidłowo. Celem utworzenia rozkazu jest jednak, aby działał on w dowolnym miejscu programu. Dlatego nie można robić założeń co do jakichkolwiek konkretnych wartości w poszczególnych rejestrach. Przyjmujemy zatem, że w rejestrach są jakieś nieznanne wartości, na których zgodnie z treścią danego rozkazu, wykonuje on odpowiednie operacje.

**Błądny przykład 6.1** (Wartość rejestru AD zdekrementować o jeden i taki wynik umieścić w akumulatorze):

```

// (AD) - 1 → AK
ROZKAZ ZS-J;
czyt wys wei il;
wyad sa weja przep weak;
wyl sa weja ode weak wea; // błąd - nieuprawnione założenie o zawartej w liczniku wartości 1

```

#### Prawidłowe rozwiązanie 6.1:

```

// (AD) - 1 → AK
ROZKAZ ZS-J;
czyt wys wei il;
wyad sa weja przep weak;
wyl sa weja dod weak wea wes il; // takt 3
wyl sa weja ode weak; // takt 4
wys sa wel; //takt 5

```

Tym razem bez względu na pierwotną wartość licznika, dodając tę wartość do akumulatora [takt 3 – *wyl sa weja dod weak*] i zwiększając jej wartość o jeden [takt 3 – *il*], a następnie odejmując od akumulatora tę zwiększoną wartość licznika [takt 4 – *wyl sa weja ode weak*] w efekcie odejmuje się od akumulatora pożądaną wartość 1.

## 4. Wnioski końcowe

W rozdziale 1 umieszczono wytyczne dotyczące przygotowania studenta do zajęć laboratoryjnych. W rozdziale 2 omówiono ogólne zagadnienia dotyczące rozwiązywania potencjalnych problemów, występujących przy implementacji rozkazów assemblerowych dla Maszyny W. W rozdziale 3 przedstawiono przykłady podstawowych błędów popełnianych przez studentów wraz z rozwiązaniami prawidłowymi. Na tej podstawie można utworzyć zbiór błędów typowych:

1. Brak prawidłowego zakończenia rozkazu - to zakończenie musi być w każdej możliwej ścieżce.
2. Utrata wartości licznika w trakcie wykonywania rozkazu - wartość licznika jest kluczowa, nie można jej 'tak po prostu' utracić.
3. Nieodpowiednie zakończenie pierwszej ścieżki instrukcji 'JEŻELI'. Należy użyć predefiniowanego słowa DALEJ... lub KONIEC.
4. Nadpisywanie zawartości rejestru I - ma on specjalne przeznaczenie i w trakcie wykonywania rozkazu, poza pierwszym taktem, nie można do niego wpisywać jakichkolwiek wartości.
5. Umieszczanie mikro sygnałów w nieodpowiednim takcie. Aby pobrać wartość z magistrali, w tym samym takcie należy ją tam umieścić. W danym takcie na magistralę można przesłać wartość tylko z jednego rejestru.
6. Nieprawdziwe, nieuprawnione wstępne założenia co do wartości danego rejestru np., że licznik zawiera wartość 1 - nie można robić takich założeń.

Artykuł ten stanowi, szczególnie dla studentów, dobre narzędzie do sprawdzenia przygotowania wstępnego do zajęć - na podstawie spisu wiedzy teoretycznej, potrzebnej do realizacji zadania tworzenia rozkazów. Artykuł ten jest także zbiorem materiału nie tylko pomocnego przy realizacji otrzymanych zadań i rozwiązywaniu potencjalnych problemów przez studentów, ale także, przy odpowiednim opanowaniu zawartych tu wskazówek, pozwala tych problemów uniknąć.

## Literatura

1. M. Chłopek, R. Tutajewicz, *Wykłady z podstaw Informatyki profesora Stefana Węgrzyna*, Skrypt uczelniany Politechniki Śląskiej nr 2062, Wydawnictwo Politechniki Śląskiej, Gliwice, 1997.
2. K. Grochła, 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.
3. A. Momot, R. Tutajewicz, *Maszyna W - jak zaprojektować prosty rozkaz*, MINUT 2019 (1), s. 24-35.
4. A. Momot, *Projektowanie rozkazów dla maszyny W - konspekt ćwiczeń laboratoryjnych*, MINUT 2020 (2), s. 1-11.
5. S. Węgrzyn, *Podstawy informatyki*, PWN, Warszawa, 1982.