Programowanie w języku asemblera

Programowanie w języku asemblera niewiele różni się od programowania w językach wysokiego poziomu. Podstawowa różnica polega na konieczności korzystania ze znacznie uboższego zestawu instrukcji. W językach asemblera brak np. instrukcji strukturalnych bądź instrukcji pozwalających w prosty sposób zapisać obliczanie wyrażenia arytmetycznego. Jednakże wszystko co możemy zapisać w językach wysokiego poziomu z pewnością można też zapisać używając instrukcji asemblera. Można także zrobić wiele działań niemożliwych do zapisania w językach wysokiego poziomu. Programista piszący w języku asemblera ma pełną kontrolę nad tym co się dzieje w trakcie obliczeń w procesorze. Nie jest on także ograniczany przez różne reguły obowiązujące w językach wysokiego poziomu. Pełna kontrola i brak ograniczeń oznaczają jednak także pełną odpowiedzialność za pisany program.

Pisząc program w języku asemblera posługujemy się niemal wyłącznie instrukcjami odpowiadającymi rozkazom danego procesora. Dlatego konieczne jest poznanie tych rozkazów, podobnie jak zaznajomienie się z architekturą tegoż procesora. Program w języku asemblera tworzą rozkazy oraz dane, na których te rozkazy działają. Program składa się z kolejnych linii, w każdej linii może znaleźć się jeden rozkaz lub deklaracja pojedynczej danej. Formalnie składnia linii programu jest następująca:

`[<etykieta>:] <rozkaz lub pseudorozkaz> [<argument>]`

gdzie:

`<etykieta>` – ciąg liter i cyfr będący symboliczną reprezentacją określonego adresu

`<rozkaz lub pseudorozkaz>` – symboliczna nazwa jednego z rozkazów procesora lub jednego z tzw. pseudorozkazów rezerwacji miejsca w pamięci na dane (RST, RPA)

`<argument>` – liczba dziesiętna lub jedna z etykiet wprowadzonych na początku linii

Pseudorozkazy RST i RPA pozwalają odpowiednio zarezerwować miejsce w pamięci na pojedynczą daną o ustalonej (jako argument) wartości początkowej oraz zarezerwować miejsce w pamięci na daną bez wskazywania jej wartości początkowej. Jako rozkaz może pojawić się nazwa jednego z dostępnych rozkazów. Przyjmiemy, że w procesorze maszyny W dostępnych jest 8 rozkazów wymienionych w poniższej tabeli.

Nazwa	Kod	Działanie
STP	000	Zatrzymanie (zakończenie) pracy programu
DOD	001	Dodanie do akumulatora zawartości komórki pamięci wskazanej przez argument
ODE	010	Odjęcie od akumulatora zawartości komórki pamięci wskazanej przez argument
POB	011	Pobranie do akumulatora zawartości komórki pamięci wskazanej przez argument
ŁAD	100	Załadowanie zawartości akumulatora do komórki pamięci wskazanej przez argument

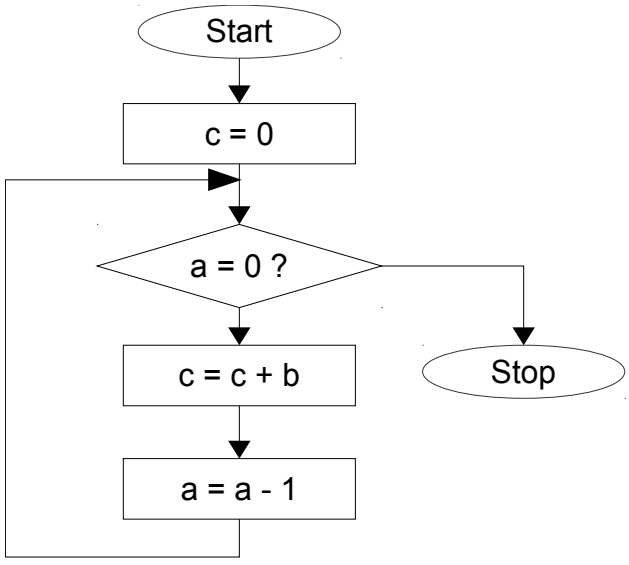
Nazwa	Kod	Działanie
SOB	101	Ustalenie, że kolejnym wykonywanym rozkazem będzie ten, który znajduje się w komórce pamięci wskazanej przez argument (tzw. skok bezwarunkowy)
SOM	110	Jeżeli w akumulatorze jest liczba ujemna, jako następny będzie wykonywany rozkaz umieszczony w komórce pamięci wskazanej przez argument. Jeżeli w akumulatorze jest liczba nieujemna, jako następny zostanie wykonany rozkaz umieszczony w pamięci bezpośrednio za rozkazem SOM
SOZ	111	Skok pod adres wskazany argumentem wykonywany tylko, gdy w akumulatorze jest 0. W przeciwnym razie jako następny zostanie wykonany rozkaz umieszczony w pamięci bezpośrednio za rozkazem SOZ

Aby napisać program w języku asemblera trzeba najpierw stworzyć algorytm rozwiązujący określone zadanie, sprecyzować go z wykorzystaniem wyłącznie dostępnych rozkazów i wreszcie zapisać go w formie linii programu w języku asemblera. Pisanie tego typów programów zilustruję kilkoma przykładami.

Przykład 1

Napisać program obliczający iloczyn dwóch liczb naturalnych *a* i *b* i umieszczający wynik w komórce oznaczonej etykietą c.

Pierwszym krokiem rozwiązania tego zadania jest wymyślenie odpowiedniego algorytmu. By znaleźć iloczyn dwóch liczb naturalnych *a* i *b*, wystarczy na wstępie założyć, że jest on równy 0 a następnie *a*-krotnie zwiększać jego wartość o *b*. Ilustruje to schemat blokowy na poniższym rysunku:



Rysunek 1. Schemat blokowy algorytmu mnożenia

Poniżej zamieszczono listing odpowiedniego programu w sytuacji, gdy zmienne a i b są równe

odpowiednio 4 i 5. Każdej operacji na schemacie zwykle będzie odpowiadać jedna lub więcej linii w tekście programu w języku asemblera. Zauważmy, że nadanie wartości początkowej zmiennej *c* nie jest reprezentowane za pomocą sekwencji rozkazów maszyny W ale zostało zrealizowane dzięki użyciu pseudorozkazu RST (linia 14 w kodzie). Testowanie, czy a przyjmuje wartość 0 realizuje kod umieszczony w liniach 1 i 2. Jeśli warunek jest spełniony ($a = 0$), należy zakończyć wykonywanie pętli, pobrać wynik do akumulatora i zakończyć program (linie 10 i 11).

```
1.          POB a
2.   pętla: SOZ koniec
3.          POB c
4.          DOD b
5.          ŁAD c
6.          POB a
7.          ODE Jeden
8.          ŁAD a
9.          SOB pętla
10.  koniec: POB c
11.         STP
12.  a:     RST 4
13.  b:     RST 5
14.  c:     RST 0
15.  Jeden: RST 1
```

Zwiększenie wartości zmiennej *c* o wartość *b* realizowane jest przez rozkazy z linii 3, 4 i 5, zaś dekrementacja zmiennej *a* to linie 6, 7 i 8. Zauważmy, że argumentem rozkazu odejmowania (linia 7) nie jest liczba 1 (`ODE 1`) ale etykieta *Jeden* wskazująca na komórkę pamięci zawierającą jedynkę (linia 15). W naszym procesorze stosowany jest tryb adresowania bezpośredniego co oznacza, że zapis `ODE 1` zostałby zinterpretowany jako chęć odjęcia od akumulatora zawartości komórki pamięci o adresie 1, tymczasem w tej komórce pamięci zostanie umieszczony rozkaz znajdujący się w drugiej linii kodu naszego programu[1]. Wreszcie w linii 9 wracamy na początek pętli do testowania warunku jej zakończenia.

Podany program można poddać asemblacji (skompilować) i, po załadowaniu do pamięci operacyjnej, wykonać. Po jego wykonaniu w akumulatorze powinna się znaleźć końcowa wartość zmiennej oznaczonej etykietą *c*, czyli w tym przypadku wartość 20.

Operacje na tablicach

Rozkazy wymienione w tabeli na stronie 1 pozwalają zapisać wiele nawet bardzo złożonych programów operujących na różnorodnych danych. Jednakże operowanie za ich pomocą na danych tworzących złożone struktury nie jest prostym zadaniem. Wymaga to wykonania dość skomplikowanych przekształceń wykorzystujących wiedzę o położeniu i organizacji danych w pamięci komputera. Problem ten zilustruję przykładem.

W pamięci komputera znajduje się *n*-elementowa tablica. Pierwszy element tej tablicy jest zapisany w komórce pamięci oznaczonej etykietą *Tablica*. Do komórki pamięci oznaczonej etykietą *Suma* należy wpisać sumę wszystkich *n* elementów tablicy.

Zanim przedstawimy rozwiązanie zadania, wyjaśnijmy jak elementy tablicy są rozmieszczone w pamięci. Tablica w pamięci komputera tworzy spójny obszar, każdy kolejny element tablicy zapisany jest w kolejnej komórce tego obszaru. Na przykład czteroelementową tablicę wypełnioną

1 Program jest zawsze ładowany do pamięci począwszy od komórki o adresie 0

czterema kolejnymi liczbami naturalnymi zapiszemy w języku asemblera maszyny W w następujący sposób:

```
1.  Tablica:  RST 1
2.            RST 2
3.            RST 3
4.            RST 4
```

Znając adres pierwszego elementu tablicy z łatwością możemy wyznaczyć położenie w pamięci (adres) któregokolwiek innego elementu tablicy. Załóżymy, że tablica, podobnie jak w języku C, będzie indeksowana od 0. Aby wyznaczyć adres w pamięci trzeciego elementu tablicy (czyli elementu o indeksie 2), należy do adresu początku tablicy dodać indeks odpowiedniego elementu. Jeżeli tablica rozpoczyna się od komórki pamięci o adresie 20, to trzeci element tej tablicy znajdziemy pod adresem 22 (20 + 2 = 22).

Aby zsumować wszystkie elementy tablicy zaczniemy od wyzerowania zmiennej *Suma*. Następnie będziemy do zmiennej *Suma* w pętli dodawać wartości kolejnych elementów poczynając od pierwszego z nich, czyli tego, który znajduje się pod adresem symbolicznym reprezentowanym etykietą *Tablica*. W każdym kolejnym przebiegu pętli będzie dodawana wartość kolejnego elementu tablicy, czyli adres dodawanego elementu za każdym przebiegiem pętli musimy zwiększać o 1. Tego typu zadanie łatwo byłoby zrealizować posługując się rozkazami używającymi adresowania pośredniego bądź indeksowego. Wiele procesorów oferuje takie rozkazy.

Na liście rozkazów naszego komputera nie znajdziemy jednak rozkazów posługujących się innymi trybami adresowania niż adresowanie bezpośrednie. Dlatego też nasze rozwiązanie będzie wymagać modyfikacji treści programu w trakcie jego działania.

```
1.  Pętla:    POB n
2.            ODE Jeden
3.            SOM Koniec
4.            ŁAD n
5.            POB Suma
6.  Rozkaz:   DOD Tablica
7.            ŁAD Suma
8.            POB Rozkaz
9.            DOD Jeden
10.           ŁAD Rozkaz
11.           SOB Pętla
12. Koniec:   POB Suma
13.           STP
14. n:        RST 4
15. Tablica:  RST 1
16.           RST 2
17.           RST 3
18.           RST 4
19. Suma:     RST 0
20. Jeden:    RST 1
```

Za każdym przebiegiem pętli trzeba modyfikować treść rozkazu dodawania kolejnego elementu tablicy, tak aby rozkaz ten rzeczywiście za każdym razem używał adresu kolejnego elementu. Skoro elementy tablicy są zapisane w pamięci po kolei jeden za drugim, oznacza to, że, jak już wcześniej

powiedziano, w każdym przebiegu pętli musimy zwiększać adres dodawanego elementu o 1. Zadanie to realizują rozkazy umieszczone w liniach 8, 9 i 10. Do akumulatora pobierany jest z pamięci rozkaz dodawania kolejnego elementu tablicy (linia 6 w programie), zwiększany o 1 i ponownie zapisywany w to samo miejsce do pamięci. Dzięki temu, gdy rozkaz ten (z linii 6) będzie ponownie wykonywany, jego argumentem będzie już kolejny element tablicy. Należy tu wyraźnie zaznaczyć, że modyfikacja z linii 8 – 10 zmienia wyłącznie treść rozkazu z linii 6 a nie powoduje zmiany znaczenia etykiety *Tablica*. Jeżeli w programie w kilku rozkazach wystąpiłaby konieczność odwoływania się do kolejnych elementów tablicy w kolejnych przebiegach pętli, wszystkie te rozkazy należałoby w podobny sposób modyfikować.

Zadanie sumowania elementów tablicy można zrealizować także nieco inaczej.

```
1.   Pętla:   POB n
2.             ODE Jeden
3.             SOM Koniec
4.             ŁAD n
5.             POB Pob0
6.             DOD Adres
7.             ŁAD Rozkaz
8.             POB Suma
9.   Rozkaz:   RPA          // tu jest rozkaz dodawania kolejnego elementu tablicy
10.            ŁAD Suma
11.            POB Adres
12.            DOD Jeden
13.            ŁAD Adres
14.            SOB Pętla
15.  Koniec:   POB Suma
16.            STP
17.  n:        RST 3
18.  Tablica:  RST 10
19.            RST -5
20.            RST 3
21.  Suma:     RST 0
22.  Jeden:    RST 1
23.  Pob0:     POB 0
24.  Adres:    RST Tablica
```

I w tym przypadku w pętli dodawane są kolejne elementy tablicy, ale nieco inaczej działa modyfikacja kodu. Rozwiązanie to przypomina rozwiązanie wykorzystujące wskaźniki w języku C. Komórka oznaczona etykietą *Adres* (linia 24) zawiera adres początku tablicy (jest jakby wskaźnikiem na tą tablicę). W każdym przebiegu pętli do dotychczasowej sumy dodawana jest wartość elementu tablicy wskazywanego przez wskaźnik *Adres* (linia 9). Aktualna wartość komórki *Adres* dodawana jest do kodu rozkazu `POB 0` (linia 23) i wynik zapisywany w miejsce oznaczone etykietą *Rozkaz* (linia 9). W ten sposób uzyskujemy rozkaz dodający do akumulatora zawartość odpowiedniej komórki pamięci, tej której adres zawiera komórka *Adres*.

Operacje wejścia/wyjścia

Dotychczasowe programy nie wymieniały informacji z otoczeniem a jedynie przetwarzały dane

na wstępie umieszczone w pamięci i tam też (w pamięci) zapisywały wyniki. Programy te działałyby znacznie bardziej ogólnie i byłyby bardziej użyteczne, gdyby dane wejściowe można było wprowadzić z klawiatury a wynik działania programu wyświetlić na ekranie. Aby to jednak było możliwe, konieczne jest uzupełnienie listy rozkazów o tzw. rozkazy wejścia/wyjścia.

Rozkazy wejścia/wyjścia

Dla zapewnienia wymiany informacji z otoczeniem należy dodać co najmniej 2 rozkazy: rozkaz wczytania pojedynczego znaku z urządzenia wejściowego (np. klawiatury) oraz rozkaz umożliwiający wyprowadzenie zawartości akumulatora na urządzenie wyjściowe (np. ekran). Argumentami obydwu tych rozkazów będzie numer urządzenia, którego dotyczy rozkaz. Przyjmiemy, że standardowe urządzenie wejściowe (klawiatura) oznaczone jest numerem 1, zaś standardowe urządzenie wyjściowe oznaczone jest numerem 2. Rozkaz wczytywania znaku nazwiemy WPR (wprowadź znak). Po wczytaniu kod wczytanego znaku zostaje umieszczony w akumulatorze. Rozkaz wyprowadzania znaku (WYP) wypisze znak, którego kod jest w akumulatorze, na urządzenie wyjściowe o podanym numerze. Wykorzystując te rozkazy można na przykład napisać program kopiujący na urządzenie wyjściowe znaki wprowadzone na klawiaturze. Program w pętli wczytuje znak i wypisuje go na standardowe wyjście. Sygnałem do zakończenia programu może być wprowadzenie jakiegoś umówionego znaku kończącego. U nas takim znakiem będzie spacja. Poniżej znajduje się kod tego programu.

```
1.      Pętla:   WPR 1
2.               ODE Spacja
3.               SOZ Koniec
4.               DOD Spacja
5.               WYP 2
6.               SOB Pętla
7.      Koniec:  STP
8.      Spacja:  RST 32
```

Zwykle programy obliczeniowe jako danych wejściowych nie potrzebują pojedynczych znaków ale liczb zapisanych w postaci ciągu cyfr. Także wyniki liczbowe należałoby wyświetlać jako ciągi cyfr dziesiętnych. Oznacza to konieczność konwersji wczytanych ciągów cyfr do postaci liczby oraz oraz odwrotnej zamiany liczby zapisanej binarnie na odpowiednią sekwencję cyfr dziesiętnych. Wydaje się, że najlepiej będzie, jeśli zadania te zostaną zrealizowane w postaci specjalnych podprogramów.

Podprogramy

Korzystanie z podprogramów oznacza konieczność dodania kolejnych kilku rozkazów do naszego procesora. Rozkaz skoku do podprogramu SDP podobnie do zwykłego skoku powoduje przejście do wykonywania rozkazu, którego adres jest argumentem rozkazu SDP. Dodatkowo jednak na stosie zapisywana jest zawartość licznika rozkazów (tzw. ślad). Dzięki zachowaniu śladu na stosie możliwy jest powrót do wykonywania następnego rozkazu po rozkazie SDP w momencie, gdy zakończono wykonywanie podprogramu. Powrót ten realizuje rozkaz PWR. Ponadto wprowadzimy jeszcze 2 rozkazy związane ze stosem: DNS – dopisujący zawartość akumulatora na szczyt stosu oraz PZS zdejmujący wartość ze szczytu stosu i zapisujący ją w akumulatorze. Rozkazy te często są używane do przekazywania parametrów między programem wywołującym a podprogramem. Poniżej przedstawiono podprogram obliczający kwadrat danej liczby. Liczba, której kwadrat chcemy znaleźć powinna najpierw zostać umieszczona w akumulatorze. Także w

akumulatorze podprogram zwróci wynik.

```
1.                SDP kwadrat    // obliczamy kwadrat liczby zapisanej w akumulatorze
2.                STP            // po wykonaniu podprogramu wynik w akumulatorze, kończymy program
3.  kwadrat:      ŁAD liczba     // tu zaczyna się podprogram, parametr zachowany w zmiennej liczba
4.                SOM ujemna     // gdy ujemna, obliczamy kwadrat z jej wartości bezwzględnej
5.                SOZ zero       // gdy zero, nic nie trzeba liczyć (wynikiem też jest zero), wracamy
6.  dodatnia:     ŁAD iloczyn    // nie ujemna i nie zero, więc dodatnia; pierwszy przebieg pętli
7.                ŁAD liczba     // potrzebne, gdy była liczba ujemna i obliczono jej wartość bezwzględną
8.  pętla:        ODE jeden      // w każdym przebiegu pętli dekrementujemy licznik wykonań
9.                SOZ wracaj     // gdy licznik osiągnie zero, kończymy
10.               ŁAD licznik    // gdy nie koniec, trzeba zapamiętać licznik
11.               POB iloczyn    // w każdym przebiegu pętli
12.               DOD liczba     // do iloczynu dodajemy liczbę, której kwadrat liczymy
13.               ŁAD iloczyn
14.               POB licznik    // a potem znów dekrementacja licznika i sprawdzenie czy nie koniec
15.               SOB pętla
16. wracaj:       POB iloczyn    // na koniec wynik w zmiennej iloczyn a powinien znaleźć się w akumulatorze
17. zero:         PWR            // i wracamy do programu głównego
18. ujemna:       ODE liczba     // parametrem liczba ujemna, a zatem obliczamy najpierw jej wartość
19.               ODE liczba     // bezwzględną, a potem postępujemy identycznie
20.               SOB dodatnia   // jak z liczbą dodatnią  (-a)^2 = a^2
21. liczba:       RPA            //  wartość parametru podprogramu
22. iloczyn:      RPA            // wynik obliczania kwadratu liczby
23. licznik:      RPA            // licznik przejść pętli (inicjowany identycznie jak liczba)
24. jeden:        RST 1
```

Program główny (linie 1 i 2) został tu uproszczony do maksimum i sprowadza się do wywołania podprogramu obliczającego kwadrat liczby zapisanej w akumulatorze[2]. Treść podprogramu rozpoczyna się w linii 3. Parametr (liczba, której kwadrat chcemy policzyć) znajduje się w akumulatorze. Najpierw podprogram sprawdza z jaką wartością parametru ma do czynienia. Jeżeli jest to zero, natychmiast przechodzimy do rozkazu powrotu z podprogramu a zwracanym wynikiem jest 0. Gdy w akumulatorze znajduje się liczba ujemna, obliczana jest liczba do niej przeciwna (lub jeśli ktoś tak woli wartość bezwzględna) a następnie postępuje się identycznie jak dla liczby dodatniej. W przypadku liczby dodatniej, by wyznaczyć jej kwadrat wystarczy liczbę tę dodać do siebie określoną liczbę razy. Zadanie to realizuje pętla, której treść zapisano w liniach od 8 do 15.

Podprogram wypisywania liczby całkowitej na urządzenie wyjściowe

Zadanie wypisywania liczby umieszczonej w akumulatorze na urządzenie wyjściowe można zrealizować w postaci podprogramu *Write* o kodzie przedstawionym poniżej.

```
1.                SDP Write
2.                STP
3.  Write:        ŁAD liczba
```

2 W celu przetestowania programu należy przed jego uruchomieniem do akumulatora wpisać liczbę całkowitą, której kwadrat chcemy obliczyć

```
4.              POB Zero
5.              DNS
6.              POB liczba
7.              SOM Abs
8.      Posit:  DZI St10
9.              MNO St10
10.             ŁAD tmp
11.             POB liczba
12.             ODE tmp
13.             DOD Znak0
14.             DNS
15.             POB tmp
16.             DZI St10
17.             SOZ Kończ
18.             ŁAD liczba
19.             SOB Posit
20.     Abs:    POB Minus
21.             WYP 2
22.             POB Zero
23.             ODE liczba
24.             ŁAD liczba
25.             SOB Posit
26.     Kończ:  PZS
27.             SOZ Wracaj
28.             WYP 2
29.             SOB Kończ
30.     Wracaj: PWR
31.     Zero:   RST 0
32.     liczba: RPA
33.     tmp:    RPA
34.     St10:   RST 10
35.     Znak0:  RST '0'
36.     Minus:  RST '-'
```

Lista rozkazów została wzbogacona o rozkazy mnożenia (MNO) i dzielenia (DZI). Podstawowym zadaniem tej procedury jest przekształcenie liczby binarnej do postaci ciągu cyfr dziesiętnych (reprezentowanych jako znaki ASCII). Konwersja ta została zrealizowana z wykorzystaniem stosu. Na stos ładowane są po kolei cyfry (zapisane z użycie kodu ASCII) będące kolejnymi resztami z dzielenia liczby binarnej przez 10. Do każdej reszty dodawany jest kod ASCII odpowiadający znakowi 0, dzięki czemu od razu dokonywana jest konwersja na znaki tego kodu. Cyfry tworzące zapis liczby generowane są zatem od końca, dzięki zastosowaniu stosu będą jednak wyprowadzane na wyjście w odwrotnej, czyli poprawnej, kolejności. W kolejnym kroku liczba zostaje podzielona przez 10 i czynności są powtarzane aż do osiągnięcia wartości 0. Wtedy ze stosu pobieramy po kolei cyfry i wypisujemy na urządzenie wyjściowe. Aby łatwo wykryć kiedy kończą się cyfry, zanim zapisywano je na stosie umieszczono tam tzw. wartownika (tu w postaci liczby 0,

zob. linie 4 i 5 w kodzie podprogramu). Pobranie ze stosu wartownika powoduje zakończenie wypisywania cyfr na wyjście i tym samym oznacza zakończenie podprogramu.

Także podczas wczytywania liczby dokonywana jest konwersja kolejnych tworzących ją znaków na odpowiednią postać binarną. Zadanie to realizuje procedura *Read*. Czyta ona kolejne cyfry tworzące zapis liczby naturalnej i konwertuje na postać binarną aż do natrafienia na znak inny niż cyfra. Po wykonaniu wczytana liczba znajduje się w akumulatorze. Kod wczytywania liczby pozostawiamy czytelnikowi do analizy.

```
1.              SDP Read
2.              STP
3.      Read:   POB Zero
4.      Wróć:   ŁAD Liczba
5.              WPR 1
6.              ODE Znak0
7.              SOM Gotowe
8.              ODE St10
9.              SOM Dalej
10.     Gotowe: POB Liczba
11.             PWR
12.     Dalej:  DOD St10
13.             ŁAD Cyfra
14.             POB Liczba
15.             MNO St10
16.             DOD Cyfra
17.             SOB Wróć
18.     Cyfra:  RPA
19.     Liczba: RPA
20.     St10:   RST 10
21.     Zero:   RST 0
22.     Znak0:  RST '0'
```

Zadania do wykonania

1. Napisać program obliczający największy wspólny dzielnik NWD dwu liczb naturalnych umieszczonych w komórkach oznaczonych etykietami A i B.
2. Napisać program obliczający najmniejszą wspólną wielokrotność NWW dwu liczb naturalnych umieszczonych w komórkach oznaczonych etykietami A i B.
3. Napisać program obliczający iloczyn dwu liczb naturalnych umieszczonych w komórkach oznaczonych etykietami A i B.
4. Napisać program obliczający resztę z dzielenia dwu liczb naturalnych umieszczonych w komórkach oznaczonych etykietami A i B.
5. Dana jest *n*-elementowa tablica oznaczona etykietą *Tab*. Napisać program sprawdzający ile razy w tej tablicy pojawia się bajt o wartości podanej w komórce *Wzorzec*.
6. Dana jest *n*-elementowa tablica oznaczona etykietą *Tab*. Napisać program zamieniający parami elementy tej tablicy. Założyć, że *n* jest liczbą parzystą. Np. jeśli w tablicy były kolejno liczby 1 2 3 i 4, to po wykonaniu programu ma być 2 1 4 3.

7. Dana jest *n*-elementowa tablica oznaczona etykietą *Tab*. Napisać program odwracający zawartość tej tablicy. Np. jeśli w tablicy były kolejno liczby 1 2 3 4 i 5, to po wykonaniu programu ma być 5 4 3 2 1.
8. Dana jest *n*-elementowa tablica oznaczona etykietą *Tab*. Napisać program znajdujący wartość największego elementu tej tablicy i wpisujący ją do komórki *Max*.