Tryby adresowania mikroprocesora Motorola 68000

Dr inż. Witold Marańda

26 października 2000

Spis treści

1 Wstęp

Termin „tryb adresowania" oznacza sposób w jaki mikroprocesor określa lokalizację operandu potrzebnego do wykonania danej instrukcji, tj. jego adres efektywny. Tryby adresowania oznacza się poprzez odpowiednią formę zapisu argumentów w poleceniach asemblera. W jednym poleceniu asemblera może wystąpić tyle różnych trybów adresowania, ile jest argumentów instrukcji.

Rozważmy następujący przykład:

```
MOVE.L    $00A00100,(A0)
```

Pierwszym argumentem jest „`$00A00100`", w którym zastosowano adresowanie bezpośrednie, a „`(A0)`" jest drugim argumentem, oznaczającym adresowanie pośrednie rejestrowe. Pierwszy operand instrukcji asemblera jest długim słowem znajdującym się pod adresem `$00A00100`, natomiast drugi operand jest adresem komórki pamięci wskazywanym przez rejestr `A0`. Pierwszy operand nazywa się „źródłowym", a drugi „przeznaczenia".

Nie wszystkie tryby adresowania mogą wystąpić ze wszystkimi instrukcjami. Ponieważ nie ma prostej do zapamiętania reguły opisującej dozwolone tryby adresowania, w razie wątpliwości należy korzystać ze szczegółowej dokumentacji poleceń mikroprocesora [1].

2 Adresowanie natychmiastowe (*Immediate addressing*)

Tryb adresowania natychmiastowego dotyczy jedynie operandów źródłowych. W trybie tym, argument polecenia asemblera będący liczbą lub symbolem poprzedzony znakiem „#" (*hash*) oznacza bezpośrednio operand biorący udział w operacji.

Przykład:

```
MOVE.B    #15,D0      *przesłanie liczby dziesiętnej 15 do D0
ANDI.B    #$EF,CCR    *skasowanie bitu X rejestru CCR
MOVEA.L   #TAB,A0     *przesłanie wartości symbolu TAB do A0
```

3 Adresowanie bezpośrednie (*Absolute/Direct addressing*)

W trybie adresowania bezpośrednim, argument może być wyłącznie liczbą lub symbolem (bez żadnych dodatkowych znaków). Argument ten jest adresem efektywnym operandu w pamięci operacyjnej (rysunek 1).

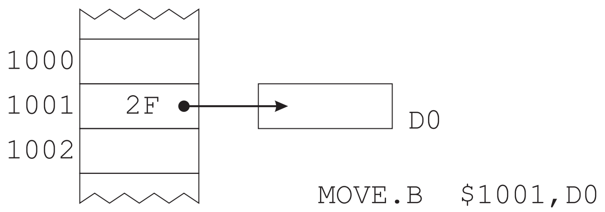
Tryb ten jest stosowany zwłaszcza przy dostępie do urządzeń wejścia/wyjścia, których położenie w przestrzeni adresowej jest znane i niezmienne.

Istnieją dwie odmiany adresowanie bezpośredniego: krótkie — jeśli adres jest liczbą 16-bitową lub długie — gdy podany adres jest 32-bitowy. Oba tryby adresowania różnią się długością generowanego kodu maszynowego.

Należy pamiętać, że w trybie adresowania krótkiego 16-bitowe adresy są przed użyciem rozszerzane do rozmiaru 32-bitów z zachowaniem znaku, więc liczby od `$8000` do `$FFFF` odpowiadają adresom od `$FFFF8000` do `$FFFFFFFF`. Oznacza to, że adresowanie bezpośrednie krótkie dotyczy jedynie pierwszych i ostatnich 32kB pamięci 32-bitowej przestrzeni adresowej.

Istnieje forma zapisu adresowania bezpośredniego, w której adres umieszcza się w nawiasach, np. `MOVE.B ($1000),D0`. Jest to forma poprawna, akceptowana przez asembler i równoważna zapisowi `MOVE.B $1000,D0`, ale w niniejszym opracowaniu używany będzie konsekwentnie zapis bez nawiasów.

Adresowanie bezpośrednie bywa też czasem nazywane, według nazwy angielskiej, adresowaniem absolutnym.



Rysunek 1: Adresowanie absolutne

Przykład:

```
MOVE.B    15,D0       *przesłanie bajtu znajdującego się pod adresem 15 do D0
AND.B     $EF,(A0)    *maskowanie bajtu maską znajdującą się pod adresem $EF
MOVEA.L   TAB,A0      *przesłanie długiego słowa spod adresu TAB do A0
```

4 Adresowanie bezpośrednie rejestrowe (*Register Direct Addressing*)

Tryb ten dotyczy operacji bezpośrednio na zawartości rejestrów mikroprocesora. Argumenty instrukcji asemblera są nazwami rejestrów wewnętrznych mikroprocesora, w których znajdują się operandy. Adresowanie rejestrowe jest najszybsze, gdyż dostęp do operandu nie wymaga dostępu do pamięci operacyjnej.

Przykład:

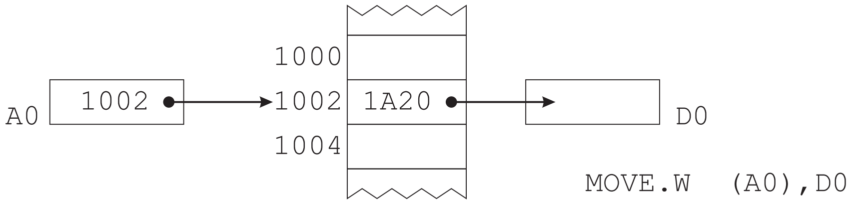
```
MOVE.L    A0,D0       *przesłanie zawartości rejestru A0 do D0
AND.B     D0,$1000    *maskowanie bajtu o adresie $1000 maską znajdującą się w D0
MOVEA.L   A1,A0       *przesłanie zawartości rejestru A1 do A0
```

Ponieważ w mikroprocesorze przyjęto generalną zasadę oddzielenia adresów od danych, operacja typu `MOVE Xn,An` jest zabroniona, podczas gdy `MOVE Xn,Dn` jest poprawna. Ma to utrudnić programiście wykonywanie nieprzemyślanych operacji na adresach. Instrukcje dotyczące adresów mają zwykle własne mnemoniki np. `MOVEA`, `ADDA` itp.

5 Adresowanie pośrednie rejestrowe (*Register Indirect Addressing*)

W trybie tym adres efektywny operandu znajduje się w rejestrze mikroprocesora (rysunek 2). Argument instrukcji asemblera ma postać „`(An)`". Do adresowania pośredniego mogą być wykorzystane wyłącznie rejestry adresowe.

Adresowanie pośrednie zapewnia szybki dostęp do danych, ponieważ ich adresy efektywne znajdują się już w rejestrach mikroprocesora i nie muszą być odczytywane z pamięci.



Rysunek 2: Adresowanie pośrednie rejestrowe

Przykład:

```
MOVE.B    #0,(A0)      *wyzerowanie bajtu, którego adres znajduje się w A0
AND.B     (A1),$1000   *maskowanie bajtu o adresie $1000 maską o adresie w A1
MOVE.L    (A2),D0      *przesłanie długiego słowa, którego adres jest w A2 do D0
```

6 Adresowanie pośrednie rejestrowe z postinkrementacją (*Register Indirect with Postincrement Addressing*)

Tryb ten jest odmianą adresowania rejestrowego pośredniego. Adres efektywny operandu znajduje się w rejestrze adresowym i po wykonaniu operacji jest zwiększany o 1, 2 lub 4 bajty, w zależności od tego jakiego rozmiaru danych dotyczyła operacja. Argument instrukcji asemblera w tym trybie ma postać „`(An)+`". Do adresowania mogą być wykorzystane wyłącznie rejestry adresowe.

Tryb adresowania z postinkrementacją jest szczególnie przydatny w sytuacji sekwencyjnego dostępu do zbiorów danych w pamięci (np. stosu), gdyż operacja obliczania adresu następnego elementu wykonuje się automatycznie i nie wymaga osobnej instrukcji.

Przykład:

```
MOVE.B    #0,(A0)+     *po przesłaniu bajtu, A0 zostaje zwiększony o 1
MOVE.L    (A2)+,D0     *po przesłaniu długiego słowa, A0 zostaje zwiększony o 4
```

7 Adresowanie pośrednie rejestrowe z predekrementacją (*Register Indirect with Predecrement Addressing*)

Tryb ten stanowi uzupełnienie trybu adresowania rejestrowego pośredniego z postinkrementacją i jest wykorzystywany w podobnych sytuacjach. Adres efektywny operandu znajduje się w rejestrze adresowym i po wykonaniu operacji jest zmniejszany o 1, 2 lub 4 bajty, w zależności od tego jakiego rozmiaru danych dotyczyła operacja.

Argument instrukcji asemblera w tym trybie ma postać „`-(An)`". Do adresowania mogą być wykorzystane wyłącznie rejestry adresowe.

Przykład:

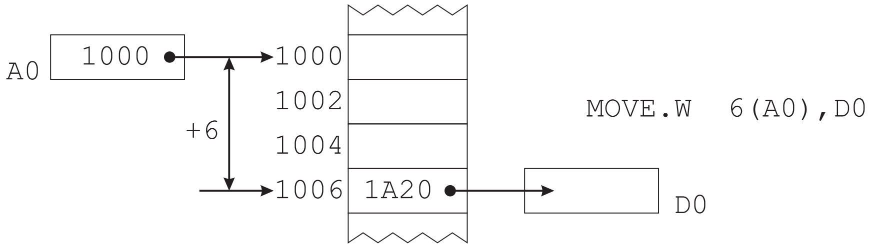
```
MOVE.W    D0,-(A0)     *po przesłaniu słowa, A0 zostaje zmniejszony o 2
MOVE.L    -(A2),D0     *po przesłaniu długiego słowa, A0 zostaje zmniejszony o 4
```

8 Adresowanie pośrednie rejestrowe z przesunięciem (*Register Indirect with Displacement Addressing*)

W trybie tym adres efektywny operandu obliczany jest poprzez sumowanie adresu znajdującego się w rejestrze z podanym przesunięciem (rysunek 3).

Argument instrukcji asemblera w tym trybie ma postać „`d`$_{16}$`(An)`". Do adresowania mogą być wykorzystane wyłącznie rejestry adresowe.

Przesunięcie jest liczbą (lub symbolem) 16-bitową, która jest rozszerzana znakowo przed dodaniem do zawartości rejestru adresowego. Oznacza to, że dostęp do operandu możliwy jest w obszarze 32kB znajdującym się za i przed komórką pamięci wskazywaną przez rejestr adresowy.



Rysunek 3: Adresowanie pośrednie rejestrowe z przesunięciem

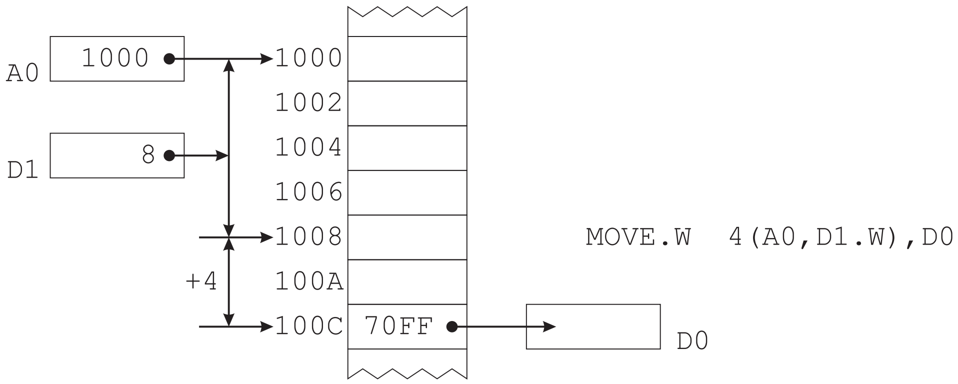
Przykład:

```
CLR.W     $10(A0)      *wyzerowanie słowa o adresie 16 bajtów powyżej adresu A0
MOVE.B    READ(A2),D0  *przesłanie bajtu, którego położenie względem A2 wynosi READ
```

9 Adresowanie pośrednie rejestrowe z indeksem (*Register Indirect with Index Addressing*)

W tym trybie, adres afektywny operandu obliczany jest przez sumowanie wartości rejestru adresowego, rejestru indeksu oraz przesunięcia (rysunek 4).

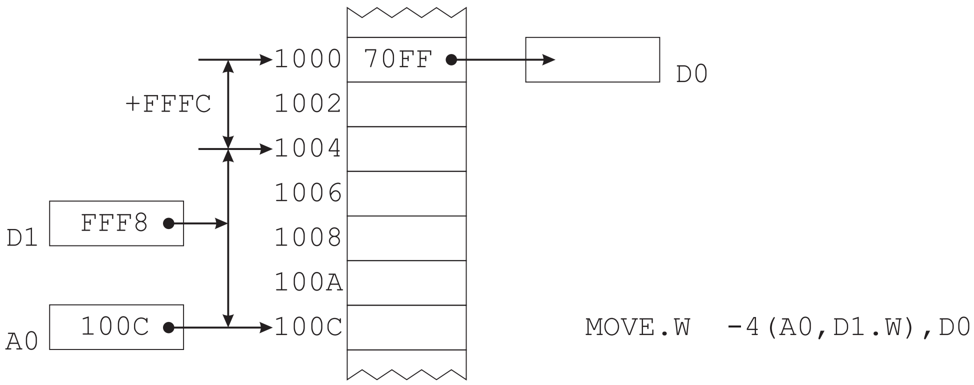
Argument instrukcji asemblera w tym trybie ma postać „`d`$_8$`(An,Xn)`". Do adresowania mogą być wykorzystane wyłącznie rejestry adresowe, natomiast rolę indeksu może pełnić każdy rejestr adresowy lub danych. Przesunięcie jest liczbą 8-bitową, rozszerzaną znakowo przed sumowaniem, dając możliwość dostępu do operandu w zakresie od -128 do $+127$, względem adresu obliczonego jako suma `An+Xn`. Rysunek 5 przedstawia sytuację gdzie zarówno indeks jak i przesunięcie są liczbami ujemnymi.



Rysunek 4: Adresowanie pośrednie rejestrowe z indeksem

Jako indeks może być wykorzystana zawartość całego 32-bitowego rejestru lub młodsza 16-bitowa część, rozszerzana znakowo przed sumowaniem. Indeks nigdy nie może mieć rozmiaru bajtu. Rozmiar indeksu należy zawsze zaznaczać specyfikatorem typu `.W` lub `.L` w celu uniknięcia niejasności.

Stosowanie trybu adresowania z przesunięciem i indeksem ułatwia dostęp do złożonych struktur danych, np. tablic dwuwymiarowych, rekordów.



Rysunek 5: Adresowanie pośrednie rejestrowe z indeksem

Przykład:

```
CLR.L    -10(A0,D0.W)      *wyzerowanie słowa o adresie A0.L+D0.W-10
MOVE.B   D0,READ(A2,A3.L)  *przesłanie bajtu do komórki o adresie A2.L+A2.L+READ
```

10 Adresowanie pośrednie licznikiem programu z przesunięciem (*Program Counter Relative with Displacement Addressing*)

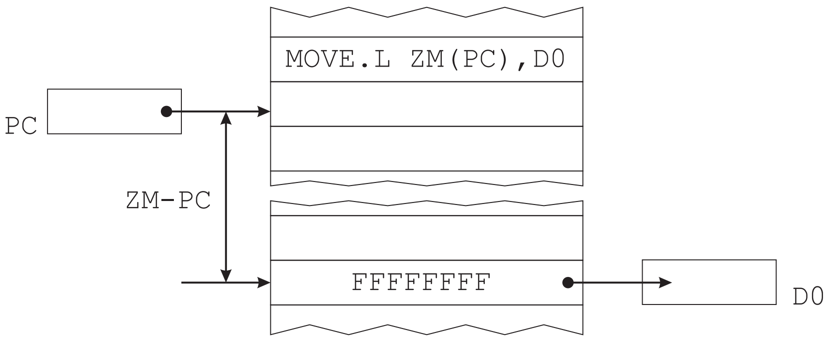
Tryb ten jest identyczny z trybem adresowania pośredniego rejestrowego z przesunięciem, z tą różnicą, że pozwala użyć licznika programu w charakterze rejestru adresowego `PC`. Adres efektywny operandu jest sumą licznika programu oraz przesunięcia.

Argument instrukcji asemblera w tym trybie ma postać „$\mathtt{d}_{16}$`(PC)`”. Przesunięcie jest liczbą (lub symbolem) 16-bitową, która jest rozszerzana znakowo przed dodaniem do zawartości rejestru.

Podstawową zaletą adresowania z wykorzystaniem rejestru `PC` jest możliwość tworzenia programów relokowalnych, tj. niezależnych od położenia w pamięci komputera. W trakcie pisania programu, względne położenia danych nie są, oczywiście, programiście znane, ale problem ten omija się posługując się etykietami. Rozważmy następujący fragment programu:

```
     ...
     MOVE.L     ZM(PC),D0
     ...
ZM   DC.L       FFFFFFFF
```

Instrukcja `MOVE` odwołuje się do komórki pamięci o adresie ZM. Asembler analizując składnię pierwszego argumentu, po napotkaniu etykiety automatycznie oblicza potrzebne przesunięcie jako różnicę `ZM-PC` i tak obliczona liczba znajdzie się w wygenerowanym kodzie maszynowym. Oczywiście, różnica pomiędzy etykietą a `PC` musi być w zakresie 32kB. Przykład ten ilustruje rysunek 6.



Rysunek 6: Adresowanie pośrednie licznikiem programu z przesunięciem

Adresowanie pośrednie z wykorzystaniem licznika programu dotyczy jedynie operandów źródłowych[1],

[1] Dostęp do pamięci w wykorzystaniem rejestru `PC` dotyczy zawsze przestrzeni programu mikroprocesora, gdzie operacje zapisu są zabronione. Uniemożliwia to (lub utrudnia) pisanie programów samomodyfikujących się.

tj. operacja `MOVE ZM(PC),D0` jest poprawna, a `MOVE D0,ZM(PC)` nie. Zatem tryb ten nadaje się do czytania zdefiniowanych stałych. W celu zapisu, adres efektywny operandu względem rejestru `PC` może być obliczany instrukcją `LEA`.

Przykład:

```
MOVE.W   TAB(PC,D0.W),D0   *przesłanie słowa z komórki o adresie PC+D0.W+(TAB-PC)
```

11 Adresowanie pośrednie licznikiem programu z indeksem (*Program Counter Relative with Index Addressing*)

Tryb ten jest rozszerzeniem adresowania względem licznika programu o możliwość stosowania indeksu. Adres efektywny operandu jest sumą licznika programu, rejestru indeksu oraz przesunięcia.

Argument instrukcji asemblera w tym trybie ma postać „$\mathtt{d}_8$`(PC,Xn)`”. Rolę indeksu może pełnić każdy rejestr adresowy lub danych, przy czym wartość indeksu może być jedynie słowem, lub długim słowem. Przesunięcie jest liczbą (lub symbolem) 8-bitową. Indeks i przesunięcie są rozszerzane znakowo przed sumowaniem.

Przykład:

```
MOVE.B   BUF(PC,A0.L),D0   *przesłanie bajtu z komórki o adresie PC+A0.L+(BUF-PC)
```

Bibliografia

[1] M68000 Family Programmer's Reference Manual, Motorola Inc., 1992, (pozycja dostępna w laboratorium)