Instrukcje mikroprocesora Motorola 68000

Dr inż. Witold Marańda

27 listopada 2000

W dokumencie skrótowo przedstawiono instrukcje mikroprocesora Motorola 68000. Omówione zostały ich funkcje i składnia oraz, tam gdzie było to istotne, przedstawiono dostępne tryby adresowania oraz wpływ instrukcji na bity rejestru `CCR`.

W celu uzyskania szczegółowych informacji dotyczącej dozwolonych trybów adresowania oraz sposobu ustawiania bitów rejestru `CCR` należy skorzystać z dokumentacji technicznej dostępnej w laboratorium [1].

W dokumencie przyjęto następujące oznaczenia:

- `Ai, Aj` — dowolny rejestr adresowy
- `Di, Dj` — dowolny rejestr danych
- `Xi, Xj` — dowolny rejestr (adresowy lub danych)
- `<ae>` — adres efektywny operandu wyrażony jednym z trybów adresowania
- `d` — stała liczbowa (opcjonalnie podano jej rozmiar w bitach np. d_{16})

1 Instrukcje przesyłu danych

`MOVE <ae>,<ae>` Instrukcja przesyła operand o rozmiarze 8, 16 lub 32 bitów. Dozwolona jest większość trybów adresowania. Instrukcja wpływa na bity rejestru `CCR`, więc np. instrukcje skoków warunkowych mogą wystapić bezpośrednio po niej, bez dodatkowych instukcji porównań.

`MOVEA <ae>,Ai` Instrukcja, jako jedyna, pozwala na przesyłanie danych do rejestrów adresowych. Bity rejestru `CCR` nie są zmieniane, gdyż operacje na adresach nie są traktowane jako operacje na danych.

`MOVE <ae>,CCR` Instrukcja pozwala na dowolne ustawianie bitów rejestru `CCR`. Operacja odczytania zawartości nie istnieje (w 68000), gdyż stan bitów `CCR` jest bezpośrednio wykorzystywany przez inne instrukcje, np. skoków warunkowych.

`MOVE <ae>,SR`, `MOVE SR,<ae>` Instrukcja pozwala na odczyt i zapis rejestru statusowego. Jest to polecenie uprzywilejowane[1] , co oznacza, że program w trybie użytkownika nie ma dostępu do rejestru`SR`.

`MOVE <ae>,USP`, `MOVE USP,<ae>` Instrukcja umożliwia dostęp do wskaźnika stosu użytkownika tj. rejestru `A7`, z programu pracującego w trybie nadzorcy.

`MOVEQ #`d_8`,Di` Instrukcja powoduje zapisanie do rejestru danych podanej wartości z zakresu od -128 do 127. Załadowana liczba jest rozszerzana do 32-bitów z zachowaniem znaku.

`MOVEP Di,`d_{16}`(Aj)`, `MOVEP `d_{16}`(Ai),Dj` Instrukcja ułatwia programowanie urządzeń peryferyjnych wyposażonych w 8-bitowe rejestry. W 16-bitowo zorientowanej pamięci systemu mikroprocesorowego, kolejne 8-bitowe rejestry urządzenia znajduja się pod adresami nieparzystymi (lub parzystymi) i nie stanowią jednolitego bloku w przestrzeni adresowej. Instrukcja `MOVEP` umożliwia jednoczesny transfer 2 lub 4 bajtów pomiędzy rejestrem danych a pamięcią tak, że adresowane są tylko bajty o adresach parzystych (lub nieparzystych).

[1] W procesorze 68000 odczytanie zawartości `SR` nie jest operacją uprzywilejowaną, natomiast we wszystkich następnych wersjach już jest.

MOVEM <lista>,<ae>, MOVEM <ae>,<lista> Instrukcja umożliwia przesłanie zawartości wielu rejestrów wymienionych na liście. Lista rejestrów ma ogólna postać Ai-Aj/Dx-Dy. Instrukcja jest zwykle wykorzystywana do odkładania i zdejmowania ze stosu rejestrów podczas pracy z podprogramami.

LEA <ae>,Ai Instrukcja oblicza adres efektywny operandu, a następnie przesyła ten adres (nie operand !) do rejestru adresowego. Rozdzielenie obliczania adresu od dostępu do operandu pozwala przyspieszyć wykonywanie programu wymagającego wielokrotne obliczanie tego samego złozonego adresu. Instrukcja pozwala również na tworzenie programów relokowalnych, przez możliwość obliczania adresów względem licznika programu. Ponadto, razem z instrukcją PEA umożliwia przekazywanie parametrów do podprogramów przez adres.

PEA <ae> Instrukcja oblicza adres efektywny operandu, a nastepnie przesyła ten adres (nie operand !) na stos. Wykorzystywana jest zwykle do obliczania adresów parametrów, które mają być przekazane do podprogramu.

EXG Xi,Xj Instrukcja wymienia zawartości dwóch dowolnych rejestrów mikroprocesora. Wymianie podlega cała 32-bitowa zawartość rejestów. W szczególności, instrukcja pozwala na transfer zawartości rejestu danych do rejestru adresowego.

SWAP Di Instrukcja zamienia miejscami zawartość starszego i młodszego słowa w rejestrze danych. Jest to przydatne zwłaszcza podczas operacji dzielenia dwóch liczb (DIVS, DIVU), a także, w przypadku programu operującego na słowach, pozwalala na „zwiększenie" liczby rejestrów do obliczeń.

2 Instrukcje arytmetyczne

ADD <ae>,Di, ADD Di,<ae> Instrukcja dodaje operand żródłowy do operandu przeznaczenia. Wynik umieszczany jest w miejscu operandu przeznaczenia. Jednym z operandów musi być rejestr danych. Dodawanie może być wykonywane na danych o rozmiarach 8, 16 lub 32 bitów i dozwolona jest większość trybów adresowania.

ADDA <ae>,Ai Instrukcja, jako jedyna, pozwala na dodawanie liczby do rejestrów adresowych. Bity rejestru CCR nie są zmieniane, gdyż operacje na adresach nie są traktowane jako operacje na danych.

ADDQ #d_3,<ae> Instrukcja dodaje liczbę od 0 do 7 do operandu o rozmiarze 8, 16 lub 32 bitów. Jest to rozszerzone polecenie inkrementacji operandu.

ADDI #d,<ae> Instrukcja dodaje liczbę o rozmiarze 8, 16 lub 32 bitów do operandu. Instrukcja, jako jedyna, pozwala na dodawanie bezpośrednio do komórek pamięci.

ADDX Di,Dj, ADDX -(Ai),-(Aj) Instrukcja oblicza sumę operandów oraz bitu X rejestru CCR, tj. wynikiem jest wartość wyrażenia $operand1 + operand2 + X$. Wynik umieszczany jest w miejscu operandu przeznaczenia. Instrukcja służy do wykonywania dodawania liczb dłuższech niż 32 bity, gdzie bit X słuzy jako przeniesienie z mniej znaczącej części. Tryb adresowania z predekrementacją pozwala wykonywać operację dodawania bezpośrednio na liczbach umieszczonych w pamięci (*big-endian*).

SUB, SUBA, SUBQ, SUBI, SUBX Instrukcje odejmowania. Uwagi na temat stosowania są analogiczne jak do instrukcji ADD.

MULS <ae>,Di, MULU <ae>,Di Instrukcja wykonuje mnożenie dwóch 16 bitowych operandów. Położenie pierwszego operandu określa tryb adresowania, a drugi znajduje się w mniej znaczacym słowie rejestru danych. 32 bitowy wynik zostaje umieszczony w rejestrze danych. Instrukcja MULS wykonuje mnożenie z uwzględnieniem znaku (*Signed*), natomiast MULU bez uwzględnienia znaku (*Unsigned*).

`DIVS <ae>,Di`, `DIVU <ae>,Di` Instrukcja wykonuje dzielenie, z uwzględnieniem znaku, 32 bitowej dzielnej znajdującej się w rejestrze danych przez 16 bitowy dzielnik, którego położenie określa tryb adresowania. 16 bitowy wynik dzielenia zostaje umiesczony w mniej znaczącym słowie rejestru danych, a 16 bitowa reszta z dzielenia - w jego bardziej znaczącym słowie. Instrukcja `DIVS` wykonuje mnożenie z uwzględnienim znaku (*Signed*), natomiast `DIVU` bez uwzględnienia znaku (*Unsigned*).

`NEG <ae>` Instrukcja powoduje negację arytmetyczną operandu, tj. wynikem jest wartość wyrażenia $0 - operand$. Operacja może dotyczyć 8, 16 lub 32 bitowych operandów.

`NEGX <ae>` Instrukcja powoduje negację arytmetyczną operandu z przeniesieniem, tj. wynikem jest wartość wyrażenia $0 - operand - X$, gdzie X jest bitem rejestru `CCR`. Operacja może dotyczyć 8, 16 lub 32 bitowych operandów.

`CLR <ae>` Instrukcja powoduje wyzerowanie operandu o rozmiarze 8, 16 lub 32 bitów. Dopuszczalna jest wiekszość trybów adresowania, w szczególności możliwe jest zerowanie bezpośrednio komórek pamięci.

`EXT Di` Instrukcja rozszerza operand, znajdujący się w rejestrze danych, o rozmiarze bajtu do rozmiaru słowa (`EXT.W Di`), lub operand o rozmiarze słowa do długiego słowa (`EXT.L Di`), z zachowaniem znaku.

3 Instrukcje arytmetyczne BCD

`ABCD Di,Dj`, `ABCD -(Ai),-(Aj)` Instrukcja oblicza sumę dwóch jedno-bajtowych liczb w kodzie BCD z uwzględnieniem przeniesienia, tj. wynikiem jest wartość wyrażenia $operand1 + operand2 + X$, gdzie X jest bitem rejestru `CCR`. Wynik umieszczany jest w miejscu operandu przeznaczenia. Uwzględnianie przeniesiania umożliwia dodawanie wielocyfrowych liczb BCD. Tryb adresowania z predekrementacją pozwala wykonywać operację bezpośrednio na liczbach umieszczonych w pamięci (*big-endian*).

`SBCD Di,Dj`, `SBCD -(Ai),-(Aj)` Instrukcja wykonuje odejmowanie dwóch jedno-bajtowych liczb w kodzie BCD z uwzględnieniem przeniesienia. Wynikiem jest wartość wyrażenia $operand2 - operand1 - X$, gdzie X jest bitem rejestru `CCR`. Uwagi na temat stosowania są analogiczne jak do instrukcji `ABCD`.

`NBCD <ae>` Instrukcja wykonuje negację arytmetyczną jedno-bajtowej liczby w kodzie BCD z uwzględnieniem przeniesienia. Wynikiem jest dopełnienie dziesiątkowe liczby gdy X=0, lub dopełnienie dziewiątkowe, gdy X=1, gdzie X jest bitem rejestru `CCR`. Uwagi na temat stosowania są analogiczne jak do instrukcji `ABCD`.

4 Instrukcje logiczne

`AND <ae>,Di`, `AND Di,<ae>` Intrukcja wykonuje operację iloczynu logicznego na odpowiadających sobie bitach operandów. Wynik umieszczany jest w miejscu operandu przeznaczenia. Jednym z operandów musi być rejestr danych. Iloczyn logiczny może być wykonywany na danych o rozmiarach 8, 16 lub 32 bity i dozwolona jest większość trybów adresowania.

`ANDI #d,<ae>` Instrukcja wykonuje operację iloczynu logicznego. Jeden z operandów wyrażony jest w postaci natychmiastowej. Wynik umieszczany jest w miejscu operandu przeznaczenia. Operacja jest mozliwa dla operandów o rozmiarach 8, 16 lub 32 bitów. Instrukcja, jako jedyna, pozwala na wykonywanie iloczynu bezpośrednio na komórkach pamięci.

`ANDI #d,CCR` Instrukcja iloczynu logicznego na zawartości rejestru `CCR`. Operand wyrażony w postaci natyczmiastowej musi mieć rozmiar bajtu.

`ANDI #d,SR` Instrukcja iloczynu logicznego na zawartości rejestru `SR`. Operand wyrażony w postaci natyczmiastowej musi mieć rozmiar bajtu. Jest to instrukcja uprzywilejowana.

`OR`, `ORI`, `ORI-CCR`, `ORI-SR` Intrukcje wykonują operację sumy logicznej na odpowiadających sobie bitach operandów. Uwagi na temat stosowania są analogiczne jak do instrukcji `AND`.

EOR, EORI, EORI-CCR, EORI-SR Intrukcje wykonują operację *exclusive or* na odpowiadających sobie bitach operandów. Uwagi na temat stosowania są analogiczne jak do instrukcji AND. Jedyną różnicą jest to, że operacja EOR <ae>,Di nie jest dopuszczalna. Instrukcja EOR Di,Di stosowana jest czasem do zerowania rejestru.

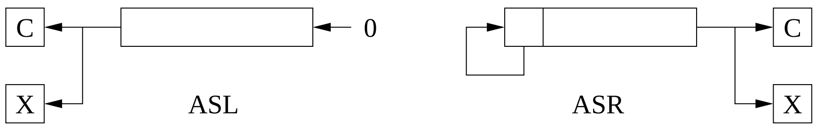
NOT <ae> Intrukcja wykonuje operację negacji logicznej operandu o rozmiarze 8, 16 lub 32 bitów. Dozwolona jest większość trybów adresowania.

5 Instrukcje przesunięć

ASL #d,Di, ASL Di,Dj, ASL <ae> Instrukcja wykonuje arytmetyczne przesunięcie bitów operandu w lewo (rysunek 1) Odpowiada to wykonaniu operacji mnożenia operandu przez 2, a bity rejestru CCR są zmieniane jak po instrukcji arytmetycznej. Bity C i X mają wartości ostatniego bitu wysuniętego z rejestru.

Liczba bitów do przesunięcia może być podana w trybie natychmiastowym (od 1 do 8), lub w rejestrze danych (od 1 do 32). Przesuwany operand, o rozmiarze 8, 16 lub 32 bitów, musi znajdować się w rejestrze danych. Jeśli podano tylko jeden argument, określający położenie operandu, to podlega on przesunięciu o 1 bit. W tym trybie możliwe jest przesuwanie bezpośrednio zawartości komórek pamięci tylko o rozmiarze słowa.

ASR #d,Di, ASR Di,Dj, ASR <ae> Instrukcja wykonuje arytmetyczne przesunięcie bitów operandu w prawo (rysunek 1) Odpowiada to wykonaniu operacji dzielenia operandu przez 2, a bity rejestru CCR są zmieniane jak po instrukcji arytmetycznej. Bity C i X mają wartości ostatniego bitu wysuniętego z rejestru. Ponieważ operacja dzielenia musi zachowywać znak liczby, więc przy przesuwaniu powielany zostaje najstarszy bit. Uwagi na temat stosowania są analogiczne jak do instrukcji ASL.

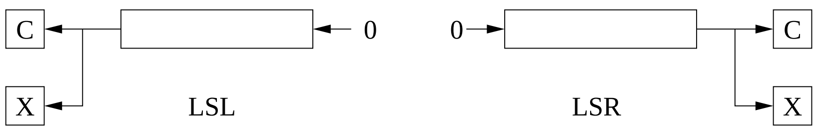


Rysunek 1: Przesunięcia arytmetyczne

LSL #d,Di, LSL Di,Dj, LSL <ae> Instrukcja wykonuje logiczne przesunięcie bitów operandu w lewo (rysunek 2). Podczas przesunięć logicznych do rejestru wsuwane są bity równe 0. Bity C i X rejestru CCR mają wartości ostatniego bitu wysuniętego z rejestru.

Liczba bitów do przesunięcia może być podana w trybie natychmiastowym (od 1 do 8), lub w rejestrze danych (od 1 do 32). Przesuwany operand, o rozmiarze 8, 16 lub 32 bitów, musi znajdować się w rejestrze danych. Jeśli podano tylko jeden argument, określający położenie operandu, to podlega on przesunięciu o 1 bit. W tym trybie możliwe jest przesuwanie bezpośrednio zawartości komórek pamięci tylko o rozmiarze słowa.

LSR #d,Di, LSR Di,Dj, LSR <ae> Instrukcja wykonuje logiczne przesunięcie bitów operandu w prawo (rysunek 2). Podczas przesunięć logicznych do rejestru wsuwane są bity równe 0. Bity C i X rejestru CCR mają wartości ostatniego bitu wysuniętego z rejestru. Uwagi na temat stosowania są analogiczne jak do instrukcji LSL.

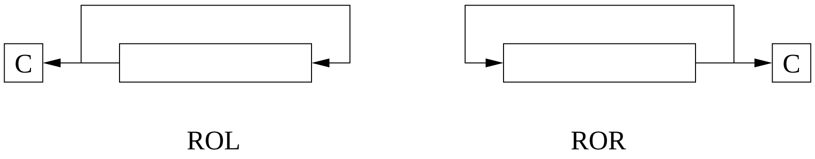


Rysunek 2: Przesunięcia logiczne

ROL #d,Di, ROL Di,Dj, ROL <ae> Instrukcja wykonuje rotacyjne przesunięcie bitów operandu w lewo (rysunek 3). Bity wysuwane z lewej strony są wsuwane z prawej strony operandu. Bit C rejestru CCR ma wartość ostatniego wysunietego bitu.

Liczba bitów do rotacji może być podana w trybie natychmiastowym (od 1 do 8), lub w rejestrze danych (od 1 do 32). Przesuwany operand, o rozmiarze 8, 16 lub 32 bitów, musi znajdować się w rejestrze danych. Jeśli podano tylko jeden argument, określający położenie operandu, to podlega on przesunięciu o 1 bit. W tym trybie możliwa jest rotacja bezpośrednio zawartości komórek pamięci tylko o rozmiarze słowa.

ROR #d,Di, ROR Di,Dj, ROR <ae> Instrukcja wykonuje rotacyjne przesunięcie bitów operandu w prawo (rysunek 3). Bity wysuwane z prawej strony są wsuwane z lewej strony operandu. Bit C rejestru CCR ma wartość ostatniego wysunietego bitu. Uwagi na temat stosowania są analogiczne jak do instrukcji ROL.

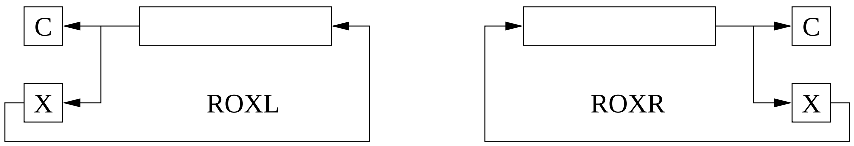


Rysunek 3: Rotacje

ROXL #d,Di, ROXL Di,Dj, ROXL <ae> Instrukcja wykonuje rotacyjne przesunięcie bitów operandu w lewo poprzez bit X (rysunek 4). Bity wysuwane z lewej strony są wsuwane do bitu X, a bit X jest wsuwany z prawej strony do operandu. Bit C rejestru CCR ma wartość ostatniego wysunietego bitu.

Liczba bitów do rotacji może być podana w trybie natychmiastowym (od 1 do 8), lub w rejestrze danych (od 1 do 32). Przesuwany operand, o rozmiarze 8, 16 lub 32 bitów, musi znajdować się w rejestrze danych. Jeśli podano tylko jeden argument, określający położenie operandu, to podlega on przesunięciu o 1 bit. W tym trybie możliwa jest rotacja bezpośrednio zawartości komórek pamięci tylko o rozmiarze slowa.

ROXR #d,Di, ROXR Di,Dj, ROXR <ae> Instrukcja wykonuje rotacyjne przesunięcie bitów operandu w prawo poprzez bit X (rysunek 4). Bity wysuwane z prawej strony są wsuwane do bitu X, a bit X jest wsuwany z lewej strony do operandu. Bit C rejestru CCR ma wartość ostatniego wysunietego bitu. Uwagi na temat stosowania są analogiczne jak do instrukcji ROXL.



Rysunek 4: Rotacje z przesunięciem

6 Instrukcje operacji bitowych

BTST #d,<ae>, BTST Di,<ae> Instrukcja testuje bit operandu i odpowiednio ustawia bit Z rejestru CCR, tj. Z=1 jeśli testowany bit jest zerem lub Z=0 w przeciwnym wypadku.

Numer bitu do testowania może być podany w formie natychmiastowej lub w rejestrze danych. Jeśli testowany operand znajduje się w rejestrze danych, to numer bitu może być od 0 do 31, a gdy znajduje się w pamięci, to testowane dotyczy tylko bajtów (bity od 0 do 7). Instrukcja tetsowanie nie wpływa na wartość testowanego operandu.

BSET #d,<ae>, BSET Di,<ae> Instrukcja testuje bit operandu identycznie jak BTST, a następnie ustawia testowany bit na 1. Uwagi na temat stosowania są analogiczne jak do instrukcji BTST.

BCLR #d,<ae>, BCLR Di,<ae> Instrukcja testuje bit operandu identycznie jak BTST, a następnie ustawia testowany bit na 0. Uwagi na temat stosowania są analogiczne jak do instrukcji BTST.

`BCHG #d,<ae>`, `BCHG Di,<ae>` Instrukcja testuje bit operandu identycznie jak `BTST`, a następnie zmienia wartość testowanego bitu na przeciwną. Uwagi na temat stosowania są analogiczne jak do instrukcji `BTST`.

7 Instrukcje porównań

`CMP <ae>,Di` Instrukcja porównuje operandy poprzez wykonanie odejmowania operandu źródłowego od operandu przeznaczenia, a następnie uaktualnienia bitów rejestru `CCR` zgodnie z wykonaną operacją. Wynik odejmowania nie jest zapamiętywany i instrukcja nie zmienia operandów. Operand przeznaczenia musi znajdować się w rejestrze danych, a dla operandu źródłowego dozwolona jest większość trybów adresowania. Porównywanie może być wykonywane na danych o rozmiarach 8, 16 lub 32 bitów.

`CMPA <ae>,Ai` Instrukcja, jako jedyna, pozwala na porównywanie z użyciem rejestrów adresowych. W przeciwieństwie do innych instrukcji operujących na rejestrach adresowych, bity rejestru `CCR` są ustawiane odpowiednio do wyniku odejmowania operandu źródłowego od operandu przeznaczenia. Wynik odejmowania nie jest zapamiętywany i instrukcja nie zmienia operandów. Porównywanie może być wykonywane tylko na danych o rozmiarach słowa lub długiego słowa. Operandy 16 bitowe są przed porównaniem rozszerzane do 32 bitów z zachowaniem znaku.

`CMPI #d,<ae>` Instrukcja porównuje operand przeznaczenia z liczbą podaną w trybie natychmiastowym. Uwagi na temat stosowania są analogiczne jak do instrukcji `CMP`. Instrukcja, jako jedyna, pozwala na porównywanie bezpośrednio wartości komórek pamięci.

`CMPM (Ai)+,(Aj+)` Instrukcja porównuje dwa operandy znajdujące się w pamięci. Uwagi na temat stosowania są analogiczne jak do instrukcji `CMP`. Jedyny dopuszczalny tryb adresowania rejestrowego pośredniego z postinkrementacją ułatwia operacje porównywania bloków pamięci.

8 Instrukcje skoków bezwarunkowych

`BRA etykieta` Instrukcja wykonuje skok bezwarunkowy względny, tj. modyfikuje rejestr `PC` tak, aby następną wykonywaną instrukcją była instrukcja oznaczona w kodzie źródłowym podaną etykietą. Instrukcja `BRA` wykonuje skok względny, tj. do rejestru `PC` dodana zostaje wartość różnicy pomiędzy adresem etykiety a aktualną wartością `PC`. Wartość tej różnicy, obliczana podczas asemblacji, zostaje zapisana w kodzie instrukcji jako liczba 8 lub 16 bitowa ze znakiem. Oznacza to, że zasięg skoku instrukcji `BRA` wynosi +32kB w przód i −32kB w tył względem jej aktualnej pozycji. Stosowanie w programie skoków względnych umożliwia pisanie programów relokowalnych.

`JMP <ae>`, `JMP etykieta` Instrukcja wykonuje skok bezwarunkowy do adresu absolutnego. Instrukcja wpisuje wartość operandu tj. adres do rejestru `PC`. Adres skoku może być podany jednym z trybów adresowania rejestrowego pośredniego z wykorzystaniem rejestrów adresowych lub rejestru `PC` albo w trybie adresowania bezpośrednim (np. z wykorzystaniem etykiety).

Skok w użyciem instrukcji `JMP` jest możliwy do dowolnego adresu w całej przestrzeni adresowej mikroprocesora. W przypadku podawania adresu skoku w trybie bezpośrednim (tj. z użyciem etykiet) w kodzie instrukcji zostaje zawarty cały 32-bitowy adres.

Programy używające skoków do etykiet są z reguły nierelokowalne. Stosowanie tego rozwiązania jest natomiast uzasadnione w przypadku odwoływania się do funkcji systemowych, których położenie w pamięci jest niezmienne, lub gdy zachodzi koniecznoć wykonania skoku dłużego niż o 32kB.

9 Instrukcje skoków warunkowych

`Bcc etykieta` Zapis `Bcc` określa grupę 16 instrukcji skoków warunkowych względnych. Instrukcja wykonuje skok do adresu określonego etykietą, jeśli spełniony jest warunek `cc`, który obliczany jest na podstawie stanu bitów rejestru `CCR`.

Instrukcja `Bcc` wykonuje skok względny, tj. do rejestru `PC` dodana zostaje wartość różnicy pomiędzy adresem etykiety a aktualną wartością `PC` (tryb adresowania pośredniego licznikiem programu z przesunięciem). Wartość tej różnicy, obliczana podczas asemblacji, zostaje zapisana w kodzie instrukcji jako liczba 8 lub 16 bitowa ze znakiem, co ogranicza zasięg skoku do +32kB w przód i −32kB w tył względem aktualnej pozycji instrukcji. Lista warunków `cc` i ich opis słowny podany jest w tabeli 1. Duża liczba warunków testowanych podczas porównywania liczb, wynika możliwości operaowania na liczbach bez znaku (`HI`, `LS`) lub ze znakiem (`PL`, `MI`, `GT`, `LT`, `GE`, `LE`).

mnemonik `cc`	opis (ang.)	warunek skoku
`CC`	Carry clear	C=0
`CS`	Carry set	C=1
`NE`	Not equal	Z=0
`EQ`	Equal	Z=1
`PL`	Plus	N=0
`MI`	Minus	N=1
`HI`	Higher then	$\overline{\mathrm{C}}\cdot\overline{\mathrm{Z}}$
`LS`	Lower than or same as	C+Z
`GT`	Greater than or equal to	$\mathrm{N}\cdot\mathrm{V}\cdot\overline{\mathrm{Z}}+\overline{\mathrm{N}}\cdot\overline{\mathrm{V}}\cdot\overline{\mathrm{Z}}=1$
`LT`	Less than	$\mathrm{N}\cdot\overline{\mathrm{V}}+\overline{\mathrm{N}}\cdot\mathrm{V}=1$
`GE`	Greater than or equal to	$\mathrm{N}\cdot\overline{\mathrm{V}}+\overline{\mathrm{N}}\cdot\mathrm{V}=1$
`LE`	Less than or equal to	$\mathrm{Z}+(\mathrm{N}\cdot\overline{\mathrm{V}}+\overline{\mathrm{N}}\cdot\mathrm{V})=1$
`VC`	Overflow clear	V=0
`VS`	Overflow set	V=1
`T`	Always true	zawsze spełniony
`F`	Always false	nigdy nie spełniony

Tablica 1: Warunki skoków instrukcji `Bcc`

`DBcc Di,etykieta` Zapis `DBcc` określa grupę 16 instrukcji wykonujacych nastepującą operację: wykonywany jest test opisany warunkiem `cc` i jeśli jest spełniony, skok nie jest wykonany (przeciwnie do `Bcc`!); jeśli warunek nie jest spełniony młodsze słowo rejestru `Di` jest zmniejszane o 1; jeśli po tej operacji wartość `Di` jest różna od -1, wykonywany jest skok do etykiety, a gdy równa -1 — skok nie jest wykonywany.

Warunki `cc` są takie same jak dla instrukcji `Bcc` (tabela 1). Instrukcja `DBcc` wykonuje skok względny w zakresie do +32kB w przód i −32kB w tył względem jej aktualnej pozycji (tryb adresowania pośredniego licznikiem programu z przesunięciem).

Działanie instrukcji `DBcc Di,etykieta` ilustruje następujacy algorytm:

```
IF (cc <> TRUE) THEN
                    BEGIN
                       Di := Di-1
                       IF (Di <> -1 ) THEN PC := etykieta
                    END
```

Instrukcje `DBcc` służą do realizacji pętli, których zakończenie zależy od dwóch kryteriów: przekroczenia zadanej ilości powtórzeń lub wystapienia sytuacji testowanej warunkiem `cc`. W szczególności, do konstrukcji pętli jedynie o zadanej ilość powtórzeń wykorzystuje się instrukcję `DBF Di,etykieta`. Maksymalna liczba powtórzeń wynosi 65535 (dekrementowane jest 16-bitowe słowo rejestru `Di`).

10 Instrukcje skoków i powrotów z podprogramów

`BSR etykieta` Instrukcja wykonuje skok do podprogramu, tj. odkłada na stos aktualną wartość rejestru `PC` (czyli 32-bitowy adres instrukcji następnej po `BSR` — adres powrotu), a następnie modyfikuje `PC` tak, aby wskazywał adres instrukcji oznaczonej w kodzie źródłowym podaną etykietą. Modyfikacja rejestru polega na dodaniu do niego 8 lub 16 bitowego przesunięcia (tryb adresowania pośredniego licznikiem programu z przesunięciem).

Instrukcja `BSR` wykonuje skok względny, tj. w zakresie do +32kB w przód i −32kB w tył względem jej aktualnej pozycji. Program napisany z wykorzystaniem `BSR` jest relokowalny.

JSR <ae>, JSR etykieta Instrukcja wykonuje skok do podprogramu, tj. odkłada na stos aktualną wartość rejestru `PC` (czyli 32-bitowy adres instrukcji następnej po `JSR` — adres powrotu), a następnie zapisuje do `PC` adres instrukcji oznaczonej w kodzie źródłowym podaną etykietą.

Instrukcja JSR wykonuje skok do adresu absolutnego, podanego w jednym z trybów adresowania rejestrowego pośredniego z wykorzystaniem rejestrów adresowych lub rejestru PC albo w trybie adresowania bezpośrednim (np. z wykorzystaniem etykiety).

Skok w użyciem instrukcji JSR jest możliwy do dowolnego adresu w całej przestrzeni adresowej mikroprocesora. W przypadku podawania adresu skoku w trybie bezpośrednim (tj. z użyciem etykiet) w kodzie instrukcji zostaje zawarty cały 32-bitowy adres.

Programy używające skoków do etykiet są nierelokowalne. Stosowanie tego rozwiązania jest natomiast uzasadnione w przypadku odwoływania się do funkcji systemowych, których położenie w pamięci jest niezmienne lub gdy podprogram znajduje się dalej niż 32kB od miejsca wywołania.

`RTS` Instrukcja wykonuje operację powrotu z prodprogramu. Nie jest wymagany żaden argument, gdyż instrukcja zdejmuje 32-bitowy adres ze stosu i ładuje do rejestru PC. Instrukcję RTS można traktować jako skok bezwarunkowy do adresu, który pobierany jest stosu.

`RTR` Instrukcja wykonuje operację powrotu z prodprogramu. Różnica w stosunku do RTS polega na tym, że przed odczytaniem ze stosu adresu powrotu, instrukcja zdejmuje ze stosu zawartość rejestru CCR. Ponieważ transfer na stos danych o rozmiarze 8-bitów nie są dozwolony, ze stosu zdjemowane jest słowo, którego młodsza część jest ładowana do CCR.

Użycie instrukcji RTR wspomaga odtworzenie zawartości CCR podczas powrotu z podprogramu. Wówczas musi być poprzedzona odłożeniem na stos aktualnej zawartości CCR na początku podprogramu, tj. `MOVE.W CCR,-(A7)`.

Instrukcję RTR mozna także wykorzystać do tworzenia podprogramów, które mogą być traktowane jako pojedyncza instrukcja, o określonej przez programistę funkcjonalności, która również odpowiednio ustawia bity rejestru CCR. Wówczas arbitralny zapis do rejestru CCR musi być ostatnią czynnością w podprogramie. Oczywiście odpowiednia wartość musi być wcześniej odłożona na stos.

11 Instrukcje dodatkowe

`Scc <ae>` Instrukcja ustawia lub zeruje wszystkie bity operandu, w zależności od wyniku testu cc. Warunki cc są opisane w tabeli 1. Jeśli warunek jest spełniony, bity operandu są ustawiane, w przeciwnym waypadku zerowane. Jedyny dopuszczalny rozmiar operandu to bajt. Po wykonaniu instrukcji wartość operandu może być wyłącznie $FF lub $00. Dozwolona jest wiekszość trybów adresowania.

`CHK <ae>,Di` Instrukcja sprawdza, czy wartość operandu w Di nie wykracza poza zakres ograniczona dwiema wartościami: 0 oraz liczbą określoną adresem efektywnym. Testowaniu podlega młodsze słowo rejestru Di. Jeśli testowany operand jest mniejszy od zera lub wiekszy od górnej granicy, wywoływana jest obsługa sytuacji wyjątkowej o numerze 6.

`NOP` Instrukcja nie wykonuje żadnej operacji. Jednym efektem działania jest ustawienia wartości rejestru PC na następną instrukcję oraz określony czas wykonywania przez mikroprocesor. Instrukcja jest czasem używana do realizacji opóźnień o określonym czasie trwania lub rezerwacji miejsca w kodzie programu na przyszłe modyfikacje np. skoki do podprogramów.

`RESET` Instrukcja powoduje ustawienie linii zewnętrznej RESET mikroprocesora w stan niski na czas 124 cykli zegarawoych. Wymusza to inicjalizację urządzeń zewnętrznych systemu mikroprocesorowego i nie ma wpływu na wewnętrzne rejestry mikroprocesora. Jest to instrukcja uprzywilejowana.

`RTE` Instrukcja powrotu z obsługi sytuacji wyjątkowej. Szczegóły działania instrukcji są przedstawione w części omawiającej sytuacje wyjątkowe. Jest to instrukcja uprzywilejowana.

STOP #d_{16} Instrukcja zapisuje 16-bitowy operand do rejestru SR, a następnie powoduje wstrzymanie przetwarzania dalszych instrukcji przez mikroprocesor. Wnowienie działania następuje jedynie w przypadku zaistnienia sytuacji wyjatkowych (trace), (interrupt) lub (reset). Jest to instrukcja uprzywilejowana.

TAS <ae> Instrukcja ustawia bity rejestru CCR w zależności od wartości testowanego operandu, a następnie zapisuje do jego najbardziej znaczącego bitu wartość 1. Jedyny dopuszczalny rozmiar operandu to bajt.

Instrukcja służy do synchronizacji procesorów w systemach wieloprocesorowych, gdyż w czasie jej wykonywania następuje cykl odczytu i zapisu, który nie może zostać przerwany przez inne urządzenie chcące przejąć kontrolę nad magistralą systemu.

TST <ae> Instrukcja porównuje wartość operandu o rozmiarze 8, 16 lub 32 bitów z liczbą zero i odpowiednio do wyniku ustawia bity rejestru CCR. Dozwolona jest większość trybów adresowania.

TRAP #d_4 Instrukcja wywoływuje obsługę sytuacji wyjątkowej *trap* o numerze od 32 do 47. Numer wektora sytuacji wyjatkowej uzystkiwany jest przez dodanie do operandu d_4 (o zakresie od 0 do 15) liczby 32.

TRAPV Instrukcja wywoływuje obsługę sytuacji wyjątkowej o numerze 7, jeśli bit V rejestru CCR jest równy 1 (*overflow*). W przeciwnym wypadku instrukcja nie wykonuje żadnej operacji.

LINK Ai,#d, UNLK Ai Instrukcje realizują mechanizm alokacji/dealokacji ramki stosu przeznaczonej na zmienne lokalne w podprogramach.

Bibliografia

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