Techniki programowania w asemblerze

Dr inż. Witold Marańda

3 stycznia 2001

Spis treści

1 Struktury programowe w asemblerze

W rozdziale przedstwiono realizacje struktur programowych typowych dla języków programowania wyższego poziomu. W przedstawionych poniżej przykładach przyjęto następujące oznaczenia:

- L — warunek logiczny: wartość operandu znajdującego się pod adresem L, która może być niezerowa (prawda) lub równa 0 (fałsz), w ogólnym przypadku, zamiast L sprawdzany może być jakikolwiek warunek,
- S, S0, S1, S2 ... — pojedyncza instrukcja lub grupa instrukcji stanowiących wykonawczą część (ciało) struktury programowej,
- N, N1, N2 — stałe liczbowe.

IF L THEN S
```
        TST     L
        BEQ     exit
        S
exit
```

IF L THEN S1 ELSE S2
```
        TST     L
        BEQ     else
        S1
        BRA     exit
else    S2
exit
```

WHILE L DO S
```
next    TST     L
        BEQ     exit
        S
        BRA     next
exit
```

REPEAT S UNTIL L
```
next    S
        TST     L
        BEQ     next
exit
```

FOR I=N1 TO N2 DO S

```
        MOVE.B  #N1,D1
next    S
        ADDQ.B  #1,D1
        CMP.B   #N2+1,D1
        BNE     next
exit
```

FOR I=N DOWNTO 0 DO S

```
        MOVE.W  #N,D1
next    S
        DBF     D1,next
exit
```

CASE L OF 0,1,2...

```
        TST     L
        BEQ     act0
        SUBI    #1,L
        BEQ     act1
        SUBI    #1,L
        BEQ     act2
        ...
other   S
        BRA     exit
act0    S0
        BRA     exit
act1    S1
        BRA     exit
act2    S2
        BRA     exit
        ...
exit
```

2 Stos

Stos jest strukturą, którą wykorzystuje się do przechowywania danych o charakterze tymczasowym. Stos znajduje się w pamięci operacyjnej komputera. Dostęp do elementów stosu odbywa się z wykorzystaniem rejestru adresowego, zwanego wskaźnikiem stosu, w którym zapisane jest położenie ostatniego elementu na stosie, zwanego wierzchołkiem stosu. Operacja dopisania (tj. odłożenia na stos) lub odczytania (tj. zdjęcia ze stosu) elementu wymaga, oprócz odczytania/zapisania wartości elementu w pamięci, zmodyfikowania wartości wskaźnika stosu, tak aby wskazywał na aktualny wierzchołek.

W systemach mikroprocesorowych opartych na procesorze 68000, systemowy stos wywołań podprogramów i przerwań ma następująca budowę:

- stos rośnie w kierunku malejących adresów, tj. każdy element ma adres mniejszy od swojego poprzednika,
- wskaźnik stosu wskazuje położenie elementu będącego wierzchołkiem stosu, a nie pierwszego wolnego adresu na stosie,
- rolę podstawowego wskaźnika stosu systemu (SP — *stack pointer*) pełni zawsze rejestr `A7`, zwany też `USP` (*user stack pointer*) w trybie użytkownika lub `SSP` (*supervisor stack pointer*) w trybie nadzorcy. Oczywiście każdy z rejestrów `A0-A6` może służyć do budowy stosu na potrzeby programu.
- operacje na stosie dozwolone są jedynie dla danych bedacych wielokrotnoscia 16-bitów, tj. slów i dlugich slów.

Realizacja stosu o wymienionych cechach wspomagana jest poprzez tryby adresowania mikroprocesora Motorola 68000. Operacje odkładania na stos realizuje się wykorzystując adresowanie rejestrowe pośrednie z predekrementacją, a zdejmowanie ze stosu — wykorzystując adresowanie rejestrowe pośrednie z postinkrementacją.

Automatyczna inkrementacja lub dekrementacja wskaźnika stosu odpowiada rozmiarowi elementu stosu, tj. transfer słowa zwiększa lub zmniejsza wartość rejestru o 2, a długiego słowa — o 4.

Przyklad

- `MOVE.W D0,-(A7)` — operacja odłożenia elementu (`D0.W`) na stos,
- `MOVEA.L (A7)+,A1` — operacja zdjęcia elementu (`A1.L`) ze stosu,
- `MOVE.L (A7)+,-(A6)` — operacja transferu długiego słowa z jednego stosu na drugi.

Za pomocą stosu możliwe jest rekurencyjne wywoływanie podprogramów, ponieważ adresy powrotu, odkładane na stosie, odczytywane są w poprawnej kolejności podczas powrotów z podprogramów. Wykorzystanie stosu nie nakłada również ograniczeń na liczbę możliwych zagnieżdżeń podprogramów (ograniczeniem jest jedynie rozmiar pamięci dostępnej dla stosu).

Stos wykorzystuje się również do przechowywania wartości zmiennych lokalnych w podprogramach oraz pośrednich wyników obliczeń.

3 Wymiana informacji z podprogramami

Wymiana informacji z podprogramem obejmuje sposoby przekazywania parametrów do podprogramów oraz wyników zwracanych przez podprogramy. Poniżej omówione zostały jedynie różne realizacje takich mechanizmów, bez klasyfikowania ich według rodzaju przekazywanej informacji (np. wartość czy adres argumentu).

3.1 Przekazywanie przez rejestry

Najprostszy i najszybszy sposób wymiany informacji z podprogramem polega na wykorzystaniu wewnętrznych rejestrów mikroprocesora. Wadą tej metody jest niewielka ilość przekazywanej informacji (najwyżej kilka rejestrów). Informacja zwrotna, oprócz zawartości rejestrów, może być też przekazywana poprzez bity rejestru `CCR`.

Poniższy przykład przedstawia wykorzystanie podprogramu `proc`, który wykonuje pewną operację na wartości najmłodszego bajtu `D0` i zapisuje do niego 0 jeśli operacja się powiodła lub numer błędu w przeciwnym wypadku. Informacja ta jest następnie analizowana przez program wywołujący:

```
        ...
        MOVE.B  parametr,D0
        BSR     proc
        TST.B   D0
        BEQ     ok
        BSR     problem
ok      ...
```

Jeśli przekazywaną daną jest adres parametru, wówczas do jego obliczenia używa się instrukcji `LEA`. W poniższym przykładzie podprogram `proc` oczekuje jako argumentu adresu w rejestrze `A0` i przy powrocie sygnalizuje błąd wykonania operacji poprzez ustawienie bitu *carry*. Użycie takiego podprogramu może wyglądać następująco:

```
        ...
        LEA     param,A0
        BSR     proc
        BCC     ok
        BSR     problem
ok      ...
```

3.2 Przekazywanie przez stos

Jeśli zachodzi potrzeba przekazania do podprogramu większej ilości danych lub gdy rejestry mikroprocesora nie mogą być do wykorzystane, najwygodniejszą metodą jest wykorzystanie obszaru stosu. Dodatkowo wykorzystanie stosu jako obszaru komunikacji z podprogramem, pozwala na rekurencyjne wykonywanie podprogramów i tworzenie programów relokowalnych.

Przed wywołaniem podprogramu, na stos odkładane są wartości i/lub adresy parametrów (instrukcja `PEA` — *Push Effective Address*). Po wykonaniu skoku, podprogram może wykorzystać te dane odwołując się do nich poprzez rejestr A7. Po powrocie, należy odtworzyć wartość wskaźnika stosu (tj. zdjąć odłożone parametry), aby wielokrotne wywołanie podprogramu nie spowodowało przepełnienia stosu[1].

Poniższy przykład przedstawia wykorzystanie podprogramu proc, który wymaga adresu parametru param1 oraz wartości parametru param2 oraz odtworzenie wskaźnika stosu po powrocie:

```
        ...
        PEA     param1
        MOVE.W  param2,-(A7)
        BSR     proc
        LEA     6(A7),A7
        ...
```

Należy pamiętać, że instrukcja skoku do podprogramu dodatkowo odłożyła na stos adres powrotu, więc odwołanie do parametrów musi uwzględniać to przesunięcie. Początek podprogramu proc, w którym dokonuje się załadowanie wartości param1 do D1 i param2 do D0 do rejestrów może mieć następującą formę:

```
proc
        MOVE.W  4(A7),D0
        MOVEA.L 6(A7),A0
        MOVE    (A0),D1
        ...
        RTS
```

Odwołanie się do wartości parametru param1 przekazywanego przez adres wymaga najpierw odczytania jego adresu ze stosu MOVEA.L 6(A7),A0, a następnie jego wartości MOVE (A0),D1. W procesorach 68020+ istnieje tryb adresowania pozwalajacy na stosowanie podwójnej referencji do operandu.

Odwołanie do parametrów w podprogramie można uprościć. W poniższym przykładzie przekazywane są do podprogramu cztery parametry. Podprogram proc może odwołać się do wartości przekazanych przez adres parametrów param3, param4 oraz przekazywanych przez wartość param1, param2, odczytując ich adresy za pomocą instrukcji przesłania grupowego MOVEM.

```
        ...
        PEA     param4
        PEA     param3
        MOVE.L  param2,-(A7)
        MOVE.L  param1,-(A7)
        BSR     proc
        LEA     16(A7),A7
        ...
proc
        MOVEM.L 4(A7),D0-D1/A0-A1
        MOVE    (A0),D2
        MOVE    (A1),D3
        ...
        RTS
```

Ponieważ kolejność transferu do rejestrów jest ogólnie D0-D7,A0-A7, to uwzględniając kolejność położenia parametrów na stosie w powyższym programie, do rejestru D0 załadowana zostanie wartość param1, do D1 wartość param2, do D2 wartość param3 oraz do D3 wartość param4.

3.3 Przekazywanie przez zmienne statyczne

Podprogramy mogą bezpośrednio odwoływać się do zmiennych lub większych struktur danych bez konieczności przekazywania ich wartości lub adresów jako parametry, o ile zmienne takie mają charakter globalny.

Odpowiada to odwoływaniu się do zmiennych globalnych w języku C, lub do bloku COMMON w języku Fortran.

[1] Procesory Intel wyposażone są w instrukcję RET n powrotu z podprogramu i automatycznego zdjęcia ze stosu określonej liczby bajtów

4 Enkapsulacja podprogramów

Realizacja fragmentów programu o określonej funkcjonalności jako podprogramów pozwala na znaczne uproszczenie zapisu programu głównego. Podprogramy mogą być wówczas traktowane jako pojedyncze instrukcje, typu `BSR proc`, realizujące złożone zadania. Jednocześnie, taka złożona instrukcja nie powinna modyfikować rejestrów mikroprocesora innych niż przeznaczone do wymiany informacji z programem wywołującym.

Wymagania te można zrealizować poprzez enkapsulację podprogramu, tj. odizolowanie od programu wywołującego. W tym celu na początku wykonywania podprogramu zachowywany jest stan rejestrów mikroprocesora, wykorzystywanych przez podprogram, których zawartość jest następnie odtwarzana przy powrocie. Najbardziej uniwersalnym miejscem do czasowego przechowywania wartości rejestrów jest stos.

W ten sposób podprogram staje się zamkniętym modułem (*black box*), komunikującym się z programem wywołującym w przewidziany sposób i nie wpływającym na inne zasoby mikroprocesora, niezależnie od stopnia swej wewnętrznej złożoności.

Enkapsulację najłatwiej uzyskać wykorzystując instrukcję przesłania grupowego:

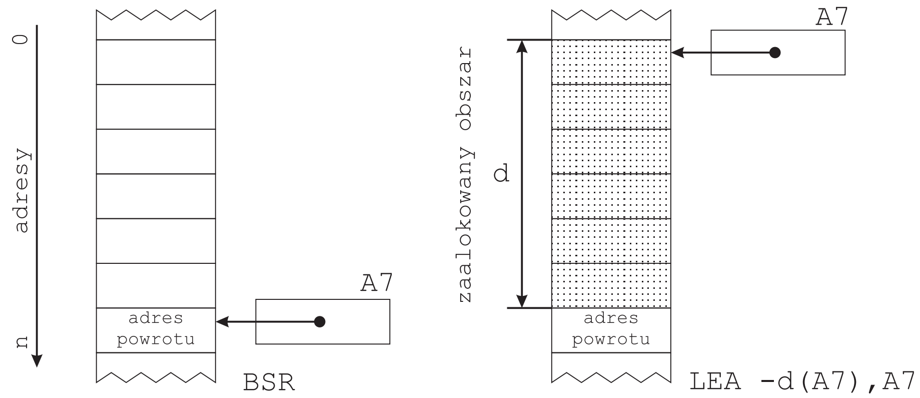
```
proc
        MOVEM.L D0-D7/A0-A6,-(A7)
        ...
        MOVEM.L (A7)+,D0-D7/A0-A6
        RTS
```

Jeśli podprogram nie powinien również zmieniać wartości rejestru `CCR`, do enkapsulacji można wykorzystać instrukcję `RTR`:

```
proc
        MOVE.W  CCR,-(A7)
        MOVEM.L D0-D7/A0-A6,-(A7)
        ...
        MOVEM.L (A7)+,D0-D7/A0-A6
        RTR
```

5 Alokacja pamięci na stosie

Bardziej złożone podprogramy potrzebują zwykle obszaru pamięci na przechowywanie pośrednich wyników obliczeń lub wartości zmiennych lokalnych. Obszar ten powinien być zwalniany po zakończeniu wykonywania podprogramu. Obszar taki nalepiej zarezerwować na stosie. Rozwiązanie takie pozwala również na rekurencyjne wywoływanie podprogramu. Alokację obszaru pamięci na stosie, przedstawia rysunek 1.



Rysunek 1: Alokacja ramki stosu

Rezerwację obszaru na stosie uzyskuje się poprzez przesunięcie wskaźnika stosu na początku podprogramu. Obszar taki musi być zwolniony przy wyjściu z podprogramu. Poniższy przykład przedstawia alokację 100 bajtów na stosie:

```
proc
```

```
        LEA     -100(A7),A7
        ...
        LEA     100(A7),A7
        RTS
```

Odwołania do zmiennych lokalnych podprogramu mogą odbywać się z wykorzystaniem wskaźnika stosu z przesunięciem np. `MOVE d(A7),D0`, lub innego rejestru, w którym zapamietany jest poczatek alokowanego obszaru, tzw. wskaźnika ramki stosu (*frame pointer* — `FP`). W poniższym przykładzie, rolę wskaźnika pełni rejestr `A0`, względem którego następuje odwołanie do wartości dwóch zmiennych lokalnych. Rozmiar zaalokowanego obszaru wynosi 8 bajtów.

```
proc
        MOVEA.L A7,A0
        LEA     -8(A7),A7
        ...
        MOVE.L  -4(A0),D0
        MOVE.L  -8(A0)
        ...
        LEA     8(A7),A7
        RTS
```

6 Ramka stosu

W ogólnym przypadku, podprogram dysponuje na stosie strukturą, na którą składają się:

- parametry (wartości i adresy) przekazywane do podprogramu,
- adres powrotu z podprogramu,
- kopia wartości rejestrów roboczych podprogramu,
- poprzednia wartość rejestru wykorzystanego jako wskaźnik ramki,
- zarezerwowany obszar pamięci na stosie, na zmienne lokalne.

Strukturę taką, która stanowi zasoby podprogramu, nazywa się ramką stosu (*stack frame*). Podprogram odwołuje się do elementów ramki stosu poprzez wskaźnik ramki, przy czym przesunięcie względem `FP` jest dodatnie dla parametrów, a ujemne dla zmiennych lokalnych.

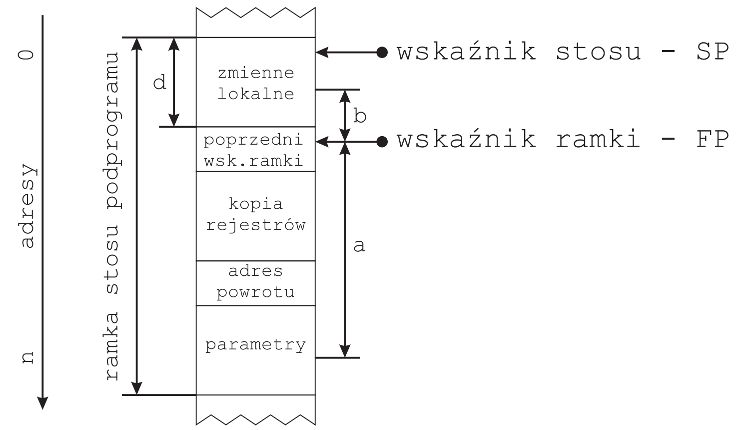
Zastosowanie wskaźnika ramki umożliwia prześledzenie historii wywołań podprogramu (*traceback*) oraz ułatwia dealokację, gdyż nie potrzeba zapamiętywać rozmiaru alokowanego obszaru.

Poniższy program przedstawia typowy początek i zakończenie podprogramu, który odwołuje się do parametrów i alokuje pamięć na zmienne lokalne. Strukturę stosu takiego podprogramu przedstawia rysunek 2.

```
proc
        MOVEM.L Xi-Xj,-(SP)     * kopia rejestrów roboczych na stosie
        MOVE.L  FP,-(SP)        * kopia bieżącej wartości FP
        MOVEA.L SP,FP           * ustawienie wskaźnika ramki stosu
        LEA     -d(SP),SP       * alokacja pamięci na stosie (d>0)
        ...
        MOVE    a(FP),Di        * odwołanie do parametru (a>0)
        MOVE    -b(FP),Dj       * odwołanie do zmiennej lokalnej (b>0)
        ...
        MOVEA.L FP,SP           * dealokacja pamięci na stosie
        MOVEA.L (SP)+,FP        * odtworzenie wartości FP
        MOVEM.L (SP)+,Xi-Xj     * odtworzenie wartości rejestrów roboczych
        RTS
```

7 Automatyczna alokacja/dealokacja — `LINK, UNLK`

Proces alokacji i zwalniania ramki stosu może być zautomatyzowany poprzez użycie instrukcji `LINK` i `UNLK`. Ułatwia to stosowanie ramek stosu o zmiennej długości i zachowuje zawartość rejestru używanego jako wskaźnik ramki, co umożliwia śledzenie historii wywołań.



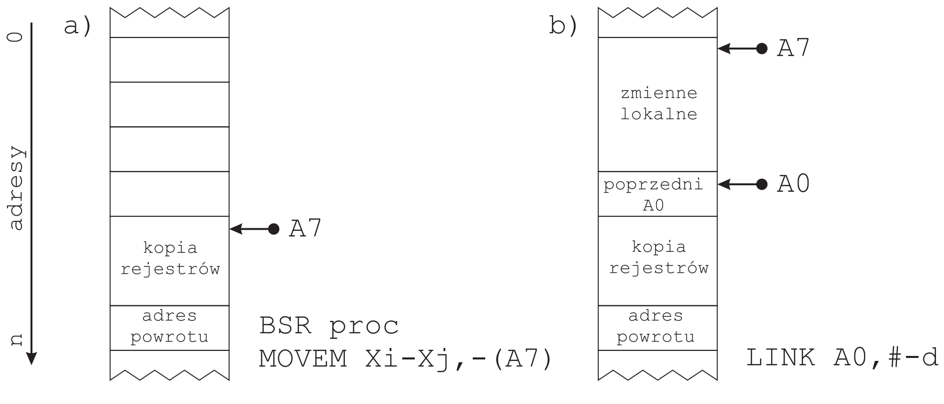
Rysunek 2: Struktura ramki stosu podprogramu

Instrukcja `LINK Ai,#-d` odkłada na stos zawartość rejestru `Ai`, zapisuje wskaznik stosu A7 w rejestrze Ai, a następnie rezerwuje obszar na stosie o rozmiarze `d`-bajtów poprzez zmniejszenie wskaznika stosu. Rejestr `Ai` staje się wskaźnikiem ramki dla danego podprogramu.

Odpowiednio, instrukcja `UNLK Ai` ładuje do wskaźnika stosu `A7` zawartość wskaźnika ramki `Ai`, tj. zwalnia zajętą pamięć, a następnie zdejmuje ze stosu i zapisuje do `Ai` jego poprzednią wartość. Na uwagę zasługuje fakt, że nie trzeba podawać rozmiaru ramki stosu podczas dealokacji.

Poniższy program ilustruje alokację ramki stosu w podprogramie. Wygląd stosu przed i po użyciu instrukcji `LINK` pokazano na rysunku 3.

```
proc
        MOVEM.L D0-D7/A0-A6,-(A7)
        LINK    A0,#-d
        ...
        UNLK    A0
        MOVEM.L (A7)+,D0-D7/A0-A6
        RTS
```



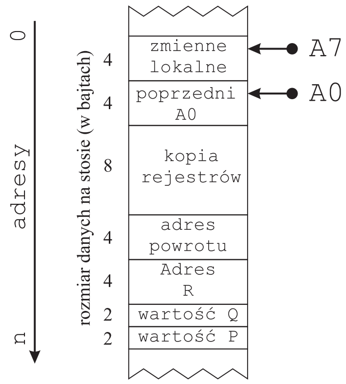
Rysunek 3: Struktura stosu po wywołaniu instrukcji `LINK`

Przykład Podprogram `sqrsum` oblicza sumę kwadratów, tj. $R = P^2 + Q^2$. Parametry P i Q są przekazywane przez wartość, natomiast R przez adres. Po wykonaniu, podprogram zmienia wartość parametru R. Podprogram odwołuje się do wyników pośrednich jak i przekazywanych parametrów z wykorzystaniem wskaźnika stosu.

```
P       DS.W    1
Q       DS.W    1
R       DS.L    1
        ...
```

```
        MOVE.W  P,-(A7)             *wartość P na stos
        MOVE.W  Q,-(A7)             *wartość Q na stos
        PEA     R                   *adres R na stos
        BSR     sqrsum              *wywołanie podprogramu
        LEA     8(A7),A7            *odtworzenie wskaźnika stosu
        ...
sqrsum
        MOVEM.L D1/A1,-(A7)         *kopia rejestrów roboczych na stosie
        LINK    A0,#-4              *rezerwacja pamięci na stosie (4 bajty)
        MOVE.W  22(A0),D1           *odczytanie wartości parametru P
        MULU    D1,D1               *obliczenie P*P
        MOVE.L  D1,-4(A0)           *zachowanie wartości P*P w alokowanym obszarze
        MOVE.W  20(A0),D1           *odczytanie wartości parametru Q
        MULU    D1,D1               *obliczenie Q*Q
        ADD.L   -4(A0),D1           *obliczenie P*P + Q*Q
        LEA     -16(A0),A1          *obliczenie położenia adresu parametru R na stosie
        MOVE.L  (A1),A1             *odczytanie adresu parametru R
        MOVE.L  D1,(A1)             *zapisanie nowej wartości do R
        UNLK    A0                  *dealokacja pamięci na stosie
        MOVEM.L (A7)+,D1/A1         *odtworzenie wartości rejestrów
        RTS
```

Struktura stosu podczas wykonywania podprogramu `sqrsum` przedstawiona jest na rysunku 4. Odwołanie do parametrów P, Q i R wymaga obliczenia przesunięcia uwzględniającego zachowane rejestry, adres powrotu oraz rozmiary przekazywanych parametrów.



Rysunek 4: Struktura stosu podczas wykonywanie podprogramu `sqrsum`