Ćwiczenie nr 4

Zasady kodowania podprogramów

4.1 Wstęp

W praktyce programowania spotykamy się często z sytuacjami, gdy identyczne czynności wykonywane są w wielu miejscach programu. W takich przypadkach tworzymy odpowiedni podprogram (w języku wysokiego poziomu nazywany często procedurą lub funkcją), który może być wywoływany w różnych miejscach programu. Poniżej rozpatrzymy szczegółowo mechanizmy wywoływania i powrotu z podprogramów na poziomie instrukcji procesora. W dalszej części opracowania przedstawimy także specyfikę wywoływania podprogramów wchodzących w skład systemu operacyjnego, nazywanych czasami ekstrakodami.

4.2 Organizacja stosu

Stos jest strukturą danych, która stanowi odpowiednik, np. stosu książek. Kolejne wartości zapisywane na stos ładowane są zawsze na jego wierzchołek. Również wartości odczytywane są zawsze z wierzchołka stosu, przy czym odczytanie wartości należy rozumieć, jako usunięcie jej ze stosu. W literaturze technicznej tak zorganizowana struktura danych nazywana jest kolejką LIFO, co stanowi skrót od ang. „Last In, First Out". Oznacza to, że obiekt który wszedł jako ostatni, jako pierwszy zostanie usunięty.

W komputerach z procesorem Pentium (i jego poprzednikach) stos umieszczany jest w pamięci operacyjnej. Obszar pamięci zajmowany przez stos określany jest jako segment stosu. Początek segmentu stosu wskazuje rejestr segmentowy SS, a położenie ostatnio zapisanej danej, czyli wierzchołek stosu, wskazuje rejestr ESP (lub SP w trybie 16-bitowym). Rejestr ESP (lub SP) nie zawiera adresu fizycznego ostatnio zapisanej danej, lecz jej odległość, obliczaną w bajtach, od początku segmentu stosu.

Wprawdzie przetwarzanie danych zapisanych na stosie może być wykonywane za pomocą wielu różnych instrukcji, to jednak kluczowe znaczenie mają dwie podstawowe instrukcje:

`push` – zapisanie danej na stosie,
`pop` – odczytanie danej ze stosu.

Podane instrukcje używane są często przechowywania zawartości rejestrów, np. instrukcje

```
push esi
push edi
```

powodują zapisanie na stos kolejno zawartości rejestrów ESI i EDI. W dalszej części programu można odtworzyć oryginalne zawartości rejestrów poprzez odczytanie ich ze stosu

```
pop  edi
pop  esi
```

W praktyce programowania stos ma jeszcze wiele innych zastosowań, m.in. służy do zapisywania zawartości wskaźnika instrukcji EIP przy wywoływaniu podprogramów. Zastosowania te omówimy w dalszej części instrukcji.

Specyficzną cechą implementacji stosu w komputerach jest przyjęcie zasady, że kolejne dane zapisywane na stosie umieszczane są w lokacjach pamięci o coraz niższych adresach, czyli

jak gdyby „wbrew prawom grawitacji”. Mówimy zwykle, że stos rośnie w kierunku malejących adresów. W trybie 32-bitowym instrukcja `push` przed zapisaniem na stosie powoduje zmniejszenie rejestru ESP o 4, natomiast instrukcja pop po odczytaniu danej zwiększa rejestr ESP o 4. Analogicznie, w trybie 16-bitowym rejestr SP jest zmniejszany lub zwiększany o 2.

4.3 Tworzenie i wywoływanie podprogramów

Rozpatrzmy prosty przykład. W komputerach PC zainstalowany jest zegar, zasilany z małej baterii, który pracuje także po wyłączeniu komputera. Za pomocą odpowiednich instrukcji można odczytać datę i czas z tego zegara, przy czym dane kodowane są w systemie BCD. Tak więc jeśli kod aktualnej godziny wpisany do rejestru AL ma postać 00100001, to oznacza że jest godzina 21. Dodajmy, że zegar podaje rok w postaci w dwóch cyfr, co było przyczyną trudności niektórych komputerów ze zmianą daty na 1.01.2000.

Wyświetlenie na ekranie aktualnego czasu i daty wymaga zamiany wartości odczytanych w kodzie BCD na znaki w kodzie ASCII. Ponieważ taka zamiana musi przeprowadzona oddzielnie dla liczby sekund, minut, godzin, dni, miesięcy i roku, warto więc zbudować podprogram, który by realizował zamianę kodu. Znaki w kodzie ASCII kodowane są na ośmiu bitach, przy czym znakom cyfr zostały przypisane kody: 0 – 30H, 1 – 31H, 2 – 32H, …, 9 – 39H. Zatem godzina 21 zapisana w postaci dwóch cyfr w kodzie ASCII będzie opisana przez dwa bajty: 00110010 00110001. Podane niżej instrukcje wykonują zamianę liczby w kodzie BCD zapisanej w rejestrze AL na dwa znaki ASCII, które zostają wpisane do rejestrów DH i DL.

```
mov   ah, al            ;przechowanie AL
shr   al, 4             ;przesunięcie wszystkich bitów rejestru AL o 4
                        ;pozycje w prawo
add   al, 30H           ;zamiana na kod ASCII
mov   dh, al            ;odesłanie starszej cyfry do rejestru DH
xchg  al, ah            ;zamiana zawartości rejestrów AL i AH
and   al, 00001111B     ;wyzerowanie 4 starszych bitów AL
add   al, 30H           ;zamiana na kod ASCII
mov   dl, al            ;odesłanie młodszej cyfry do rejestru DL
```

Jeśli gdziekolwiek w programie zajdzie konieczność zamiany kodu BCD na ASCII, to wystarczy tylko wywołać podaną sekwencję instrukcji. Można to zrealizować, np. za pomocą zwykłej instrukcji skoku bezwarunkowego. Taką instrukcję skoku można by też umieścić na końcu powyższej sekwencji, by procesor powrócił do kontynuowania wykonywania programu w miejscu, gdzie wywołano ww. instrukcje. Byłoby to jednak możliwe tylko wówczas, gdyby wywołanie następowało zawsze z tego samego miejsca programu. Jeśli wywołanie następuje z różnych miejsc programu, to w każdym przypadku należałoby wykonać skok do innego miejsca.

Powyższe rozważania prowadzą do wniosku, że wywołanie ciągu instrukcji tworzącego podprogram wymaga wykonania nie tylko skoku, ale przekazania także informacji dokąd należy wrócić po wykonaniu tego ciągu. Innymi słowy, trzeba podać liczbę, która ma zostać wpisana do wskaźnika instrukcji EIP po zakończeniu wykonywania podanej sekwencji instrukcji.

Jeśli 3-bajtowa instrukcja wywołująca podprogram znajduje się w komórkach pamięci o adresach 715, 716 i 717, to po zakończeniu wykonywania podprogramu powinna zostać wywołana kolejna instrukcja, czyli po zakończeniu wykonywania podprogramu zawartość rejestru EIP powinna wynosić 718. Liczba 718 musi więc być przekazana do podprogramu. W procesorach Pentium liczba ta, nazywana śladem, wpisywana jest na wierzchołek stosu. Instrukcja skoku bezwarunkowego, która przekazuje sterowanie do podprogramu (tj. ustawia

EIP, tak by wskazywał adres komórki pamięci zawierającej pierwszy rozkaz podprogramu) i dodatkowo zapamiętuje ślad na stosie w wielu procesorach nosi nazwę `call`.
Z kolei na końcu podprogramu należy umieścić instrukcję, która przepisze liczbę ze stosu do wskaźnika instrukcji EIP – instrukcja taka nosi nazwę `ret`.

W asemblerze początek podprogramu wskazuje dyrektywa `PROC`, a koniec podprogramu dyrektywa `ENDP`. Dyrektywy te nie generują żadnych dodatkowych instrukcji, ale jedynie polepszają czytelność programu.

```
BCD2ASCII   PROC  near
      mov   ah, al            ;przechowanie AL
      shr   al, 4             ;przesunięcie wszystkich bitów rejestru
                              ;AL o 4 pozycje w prawo
      add   al, 30H           ;zamiana na kod ASCII
      mov   dh, al            ;odesłanie starszej cyfry do rejestru DH
      xchg  al, ah            ;zamiana zawartości rejestrów AL i AH
      and   al, 00001111B     ;wyzerowanie 4 starszych bitów AL
      add   al, 30H           ;zamiana na kod ASCII
      mov   dl, al            ;odesłanie młodszej cyfry do rejestru DL
      ret
BCD2ASCII   ENDP
```

Podprogram ten może być wywoływany z dowolnego miejsca w programie za pomocą instrukcji

```
call  BCD2ASCII
```

Instrukcja `call` ma dwie odmiany:

- typu NEAR, jeśli sterowanie przekazywane jest do podpogramu znajdującego się w tym samym segmencie (CS = const);
- typu FAR, jeśli sterowanie przekazywane jest do podprogramu w innym segmencie (co wymaga zmiany zawartości rejestru CS).

Obie odmiany mogą być używane zarówno w trybie 16- jak i 32-bitowym, jednak instrukcja `call` typu far w trybie 32-bitowym praktycznie nie jest używana.

Instrukcje `call` można także podzielić ze względu na sposób określania adresu wywoływanej procedury:

- typu bezpośredniego, jeśli adres wywoływanej procedury (typu NEAR lub FAR) stanowi część kodu instrukcji;
- typu pośredniego, jeśli adres wywoływanej procedury zawarty jest w lokacji pamięci wskazywanej przez adres podany w instrukcji `call`.

W procesorach Pentium stosowana jest konwencja przechowywania liczb znana jako mniejsze niżej; w szczególności przyjmuje się, że część segmentowa adresu stanowi w pewnym sensie starszą część adresu, a więc zajmuje miejsce w pamięci o wyższym adresie; zatem jeśli na stos ładowany jest adres typu FAR, to najpierw ładowana jest część segmentowa, a następnie przesunięcie (offset).

W chwili zakończenia wykonywania podprogramu ślad zapamiętany na stosie wpisywany jest do wskaźnika instrukcji EIP. Jeśli podprogram został wywołany za pomocą instrukcji `call` typu FAR, to ślad zawiera także liczbę, która powinna zostać przepisana do rejestru CS. Z tego powodu zdefiniowano dwie odmiany instrukcji `ret`: instrukcja `retn` przepisuje liczbę z wierzchołka stosu do rejestru EIP, zaś instrukcja `retf` przepisuje dodatkowo następną liczbę ze stosu do rejestru CS. W niektórych asemblerach używany jest

tylko mnemonik `ret`, a typ instrukcji jest określany na podstawie parametru podanego po dyrektywie PROC.

4.4 Podprogramy systemowe (ekstrakody)

Działania wykonywane w komputerach są sterowane i nadzorowane przez systemy operacyjne. System operacyjny, w oparciu o polecenie otrzymywane od użytkownika stara się zapewnić możliwie najbardziej efektywną pracę komputera. W skład systemu operacyjnego wchodzą także rozmaite podprogramy usługowe, które mogą być wywoływane ze zwykłego programu. Ponieważ podprogramy te wchodzą w skład systemu operacyjnego, przyjęto nazywać je podprogramami systemowymi. Znaczna część tych podprogramów udostępnia rozmaite operacje związane z urządzeniami komputera, jak np. wyświetlenie znaku na ekranie, odczytanie znaku napisanego na klawiaturze czy też operacje zapisu i odczytu danych na dysku.

Nasuwa się pytanie czy zwykły program w celu komunikacji z urządzeniami zewnętrznymi powinien posługiwać się podprogramami systemowymi czy też może realizować taką komunikację samodzielnie. Odpowiedź na to pytanie zależy od rodzaju systemu komputerowego, stopnia jego złożoności a przede wszystkim od typu zainstalowanego systemu operacyjnego.

Właściwe sterowanie urządzeniami komputera wymaga znajomości ich budowy wewnętrznej, natomiast niewłaściwe sterowanie urządzeniem, np. drukarką może doprowadzić do jego przedwczesnego zużycia lub nawet uszkodzenia. Podprogram systemowy, opracowany przez specjalistów, będzie z pewnością sterował urządzeniem możliwie jak najlepiej. Z tego względu w wielu systemach operacyjnych wprowadzono obowiązkowe pośrednictwo podprogramów systemowych, co oznacza, że jakakolwiek próba bezpośredniego sterowania urządzeniem spowoduje przerwanie wykonywania programu.

Dodatkowym argumentem przemawiającym za stosowaniem podprogramów systemowych jest konieczność wirtualizacji urządzeń w środowisku wielozadaniowym, np. w systemie Windows. W takim środowisku może być jednocześnie wykonywanych wiele programów i mogą pojawiać się konflikty w zakresie dostępu do urządzeń. Aby temu zapobiec, we współczesnych systemach operacyjnych tworzy się urządzenia wirtualne (nierzeczywiste), które udostępnia się aktualnie wykonywanym programom. Oznacza to, że program może żądać przydzielenia drukarki i kierować do niej swoje wyniki, aczkolwiek w tym samym czasie rzeczywista drukarka może drukować wyniki innego programu. Dane skierowane do drukarki wirtualnej przechowywane są tymczasowo na dysku i później mogą być wydrukowane. W systemach wielozadaniowych możliwość bezpośredniego sterowania drukarką i innymi urządzeniami mogłaby szybko doprowadzić do dezorganizacji całego systemu.

Wyżej wymienione argumenty są mniej istotne w systemach jednozadaniowych. Przykładowo, w systemie DOS opracowano szereg programów systemowych, ale jednocześnie nie zablokowano możliwości bezpośredniego sterowania urządzeniami. Tym samym środowisko DOSu może być w pewnych przypadkach bardziej odpowiednie do przeprowadzenia różnych testów urządzeń. Można powiedzieć, że podprogramy systemowe wykonują pewne zlecenia czy usługi dla zwykłych programów użytkownika. Stąd też nazywa się je podprogramami usługowymi czy też funkcjami usługowymi systemu. Opisy podprogramów systemowych tworzą interfejs aplikacji użytkownika, nazywany w skrócie API (ang. *Application Program Interface*). Opisy takie dla poszczególnych systemów operacyjnych dostępne są w wielu dokumentach, książkach a także na stronach WWW. Przykładowo, opis API dla systemu DOS zawarty jest m.in. w książce Bułhaka „DOS od środka". Opisy funkcji systemowych, które udostępniane są w 32-bitowych wersjach systemu Windows tworzą „Win32 API" – szczegółowe opisy tych funkcji wraz z przykładami zawarte

są m.in. w plikach pomocy („Help”) dostępnych w wielu środowiskach programowania (np. Borland C++).

4.5 Wywoływanie podprogramów systemowych

Wywoływanie podprogramów systemowych mogłoby być w zasadzie wykonywane w taki sposób jak wywoływanie zwykłych podprogramów. Oczywiście podprogramy systemowe musiałaby znajdować się w ustalonych obszarach pamięci, a adresy tych obszarów powinny stanowić część dokumentacji API. Takie rozwiązanie jest jednak niepraktyczne, ponieważ systemy operacyjne nieustannie się rozwijają, powstają ich nowe wersje a zawarte w nich podprogramy systemowe są stale ulepszane i rozszerzane. Zatem położenia podprogramów w pamięci komputera też ulegają zmianom. Trzeba by więc każdy program dostosowywać do konkretnej wersji systemu operacyjnego, co oczywiście jest nie do przyjęcia.

W tej sytuacji wybrano rozwiązanie, w którym wywoływanie podprogramów systemowych odbywa się za pośrednictwem tablicy adresowej. W tablicy tej zapisane są aktualne adresy wszystkich podprogramów systemowych, a w przypadku zainstalowania nowej wersji systemu adresy te są odpowiednio aktualizowane. W celu wywoływania podprogramu systemowego nie podaje się jego położenia w pamięci operacyjnej, ale podaje się indeks do tablicy adresowej. Zatem sposób wywoływania podprogramów systemowych nie będzie ulegał zmianie po zainstalowaniu nowszej wersji systemu operacyjnego.

W procesorach Pentium omawiana tablica adresowa używana jest także do obsługi przerwań sprzętowych i wyjątków procesora, i z tego powodu nazywana jest:

- w trybie rzeczywistym (16-bitowym) i V86 – tablicą wektorów przerwań., zaś
- w trybie chronionym – tablicą deskryptorów przerwań.

Ponieważ zasady funkcjonowania procesora w trybie chronionym są znacznie bardziej złożone niż w trybie rzeczywistym, więc zajmiemy się tutaj głównie tablicą wektorów przerwań. Ogólne zasady wykorzystania tej tablicy odnoszą się także do tablicy deskryptorów przerwań.

Tablica wektorów przerwań umieszczona jest w obszarze pamięci o adresach fizycznych 00000H-003FFH (pierwszy kilobajt pamięci) i zawiera 256 adresów – adres zapisany jest w postaci segment:offset i zajmuje cztery bajty.

Instrukcja `int` została specjalnie zaprojektowana do wywoływania podprogramów za pośrednictwem tablicy wektorów przerwań. W polu operandu tej instrukcji podaje indeks do tablicy adresowej w postaci liczby z przedziału 0–255, często podawanej w postaci szesnastkowej, np. `int 21H`. Wykonanie instrukcji powoduje zapamiętanie śladu na stosie, a następnie rozpoczęcie wykonywania podprogramu, którego adres podany jest na wskazanej pozycji tablicy wektorów przerwań.

Ponieważ ślad instrukcji `int` zawiera rejestr IP, rejestr CS oraz rejestr znaczników (zob. rysunek), więc w tym przypadku w celu powrotu z podprogramu używana jest instrukcja `iret`.

Na ogół w systemie operacyjnym udostępnionych jest co najmniej kilkadziesiąt, a czasami nawet kilkaset funkcji usługowych. Tablica wektorów przerwań zawiera jednak tylko 256 adresów, co jest niewystarczające. Z tego powodu podprogramy systemowe grupowane są w pakiety. W takim przypadku adres zawarty w tablicy wektorów przerwań zawiera adres pakietu podprogramów. Bezpośrednio przed wywołaniem pakietu podprogramów trzeba wskazać konkretny podprogram poprzez podanie jego numeru. Numer podprogramu w pakiecie podaje się zwykle w rejestrze AH.

Rozpatrzmy dla przykładu funkcję wyświetlania znaku na ekranie udostępnianą przez system DOS. Pakiet funkcji usługowych systemu DOS wskazany jest przez adres zawarty w wektorze 33 (21H). Funkcja DOSu nr 2 powoduje wyświetlenie na ekranie znaku ASCII,

którego kod znajduje się w rejestrze DL. Zatem w celu wyświetlenia na ekranie dwóch znaków ASCII zawartych w rejestrach DH i DL należy wykonać poniższe instrukcje

```
xchg  dh, dl      ;zamiana zawartości rejestrów DH i DL
mov   ah, 2       ;wpisanie numeru funkcji DOSu do rejestru AH
int   21H         ;wyświetlenie znaku
xchg  dh, dl
mov   ah, 2       ;wpisanie numeru funkcji DOSu do rejestru AH
int   21H         ;wyświetlenie znaku
```

Przy wywoływaniu funkcji systemowych potrzebne parametry wpisuje się zazwyczaj do odpowiednich rejestrów. Po wykonaniu podprogramu zawartości rejestrów pozostają na ogół niezmienione, chyba że rejestr służy do wyprowadzania wyników. Znacznik CF używany jest przez wiele funkcji do sygnalizacji błędu wykonania – wówczas CF = 1, a do rejestru AX zostaje wpisany kod błędu.

Jak już wspomnieliśmy, opisy funkcji systemowych zawarte są w wielu publikacjach. W niektórych z nich funkcje usługowe systemu DOS określane są niezbyt trafnie jako „przerwania DOSu". Rozpatrzmy bliżej tę kwestię.

Analiza działania instrukcji `int` pozwala stwierdzić, że instrukcja ta wykonuje czynności charakterystyczne dla instrukcji wywołujących podprogramy: zapamiętanie śladu na stosie i przekazanie sterowania (skok) do podprogramu. Stanowi ona więc odmianę instrukcji `call` typu FAR z adresowaniem pośrednim. W pewnych sytuacjach, gdy użycie instrukcji `int` jest niewskazane, zastępuje się ją instrukcją `call` typu FAR poprzedzoną dodatkowo instrukcją `pushf` (ażeby ślad na stosie miał taką samą postać jak przy oryginalnym `int`).

Z drugiej strony instrukcja `int` wykazuje też pewne podobieństwo do funkcji przerwań sprzętowych: korzysta z tablicy wektorów przerwań, a struktura śladu zapamiętywanego na stosie jest identyczna jak przy przerwaniu sprzętowym. Zarówno instrukcja `int` jak i przerwanie sprzętowe powodują wyzerowanie znacznika IF. Ponadto mnemonik `int` stanowi skrót angielskiego słowa *interrupt* – przerwanie. Z tych powodów podprogramy systemowe wywoływane za pomocą instrukcji `int` nazywa się często i nieściśle przerwaniami. Taka terminologia zamazuje różnice między przerwaniami sprzętowymi a zleceniami wysyłanymi przez program za pomocą instrukcji `int`. W odniesieniu do procedur obsługujących zlecenia wysyłane przez program lepiej używać terminu „funkcje (lub usługi) systemowe" albo „przerwania programowe".

W środowisku systemu DOS procedury systemowe nie są chronione w szczególny sposób i mogą być także wywoływane niestandardowo. Inna sytuacja występuje w komputerach pracujących wieloprogramowo, gdzie konieczna jest ochrona procedur systemowych przez przypadkowym lub zamierzonym uszkodzeniem przez program użytkownika. W takich komputerach (dotyczy to także procesorów Pentium pracujących w trybie chronionym) podprogramy systemu operacyjnego mają znacznie większe uprawnienia niż zwykłe programy. Z tego powodu konieczna jest automatyczna zmiana uprzywilejowania w chwili wywołania procedury systemowej. Taka zmiana uprzywilejowania następuje także po przyjęciu przerwania sprzętowego lub po wykonaniu instrukcji `int`. Oznacza to, że w tego systemach instrukcja wywołująca procedurę systemową nie może być zastąpiona przez inne instrukcje o podobnym działaniu.

W przeszłości używano także terminu ekstrakod w odniesieniu do instrukcji wywołujących podprogramy systemowe – termin ten, obecnie rzadko używany, pozwalał na wyraźne odróżnienie wywołania podprogramu systemowego od przerwania sprzętowego. Podane tu rozważania nie zmieniają jednak faktu, że „Przerwania programowe ... są to w rzeczywistości przerwania grzecznościowe ... które niczego nie przerywają". (J. Duntemann, „Zrozumieć asembler", 1993).