Laboratorium Architektury Komputerów

Ćwiczenie 3

Konwersja liczb binarnych

Komputery wykonują operacje przetwarzania danych na wartościach binarnych, podczas gdy współczesna cywilizacja posługuje się systemem dziesiętnym. Zachodzi więc potrzeba dokonywania konwersji dziesiętno-dwójkowej lub dwójkowo-dziesiętnej. Nie dotyczy to jednak wszystkich danych: niektóre rodzaje danych wygodniej jest interpretować, jeśli zostaną przekształcone na postać w systemie szesnastkowym. Poniżej opisano cztery rodzaje konwersji podstawy liczb za pomocą programów kodowanych na poziomie rozkazów procesora.

Konwersja dwójkowo–dziesiętna

Obliczenia realizowane przez procesor wykonywane są na liczbach binarnych w różnych formatach. Podstawowe znaczenie mają jednak liczby kodowane w naturalnym kodzie binarnym (NKB), które w dokumentacji procesora określane są jako *liczby całkowite bez znaku*. Istnieje kilka metod przekształcenia liczb binarnych na postać dziesiętną. Opisana poniżej metoda polega wielokrotnym dzieleniu liczby przez 10.

W celu wyjaśnienia metody weźmy pod uwagę liczbę dziesiętną 5804. W celu uzyskania cyfr dziesiętnych tej liczby wystarczy wielokrotnie dzielić ją przez 10 i rejestrować uzyskiwane reszty, tak jak pokazano w podanym obok przykładzie. Najpierw uzyskano resztę 4, która stanowi liczbę jedności, potem resztę 0, która stanowi liczbę dziesiątek, i następnie resztę 8 – liczba setek, itd.

5804	4
580	0
58	8
5	5
0	

Ten sam algorytm można zastosować w odniesieniu do liczb binarnych — wartość ilorazu i reszty z dzielenia nie zależy bowiem od podstawy systemu liczbowego. Ilorazy i reszty uzyskane w trakcie wielokrotnego dzielenia liczby binarnej 0001 0110 1010 1100 = $(5804)_{10}$ przez 10 podano w poniższej tabeli.

Kolejne ilorazy z dzielenia przez 10, tj. dzielenia przez $(0000\ 1010)_2$	**Kolejne reszty z dzielenia przez 10 tj. dzielenia przez $(0000\ 1010)_2$**
0000 0010 0100 0100 (= 580)	0100 (= 4)
0000 0000 0011 1010 (= 58)	0000 (= 0)
0000 0000 0000 0101 (= 5)	1000 (= 8)
0000 0000 0000 0000 (= 0)	0101 (= 5)

Reszty z kolejnych dzieleń są wartościami cyfr jedności, dziesiątek, setek, tysięcy, itd. przetwarzanej liczby.[1] Zatem w celu wyświetlenia na ekranie liczby w postaci dziesiętnej

[1] Warto dodać, że stosowana tu metoda konwersji nadaje się także dla systemów o podstawie innej niż 10. W kolejnych krokach dzielimy daną liczbę przed podstawę systemu docelowego, np. przez 8 – otrzymywane reszty są kolejnymi cyframi rozwinięcia pozycyjnego w systemie ósemkowym.

wystarczy tylko zamienić uzyskane reszty na kody cyfr w kodzie ASCII, odwracając przy tym kolejność cyfr. Opisana metoda stosowana jest w podanym dalej fragmencie programu, w którym przeprowadzana jest konwersja liczby binarnej bez znaku zawartej w rejestrze EAX na postać dziesiętną.

Na liście rozkazów procesorów rodziny x86 dostępne są dwa rozkazy dzielenia liczb całkowitych: DIV do dzielenia liczb bez znaku i IDIV do dzielenia liczb całkowitych ze znakiem w kodzie U2. Oba te rozkazy wymagają podania położenia dzielnika w polu operandu, nie określa się natomiast położenia dzielnej, ponieważ znajduje się ona zawsze w ustalonym rejestrze lub rejestrach (zob. tablica podana niżej).

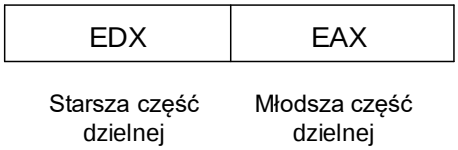
W podanym dalej fragmencie programu dzielenie wykonywane jest za pomocą rozkazu DIV. Rozkaz ten wykonuje dzielenie dwóch liczb całkowitych bez znaku, przy czym liczba bitów dzielnej jest dwukrotnie większa od liczby bitów dzielnika. W wyniku dzielenia uzyskuje się iloraz i resztę — spełniony jest związek:

```
dzielna = iloraz * dzielnik + reszta    (reszta < dzielnik)
```

Dzielnik podany jest w polu argumentu rozkaz DIV, natomiast dzielna, iloraz i reszta znajdują się w ustalonych rejestrach, tak jak podano w tabelce:

Liczba bitów		Położenie dzielnej	Położenie dzielnika	Położenie ilorazu	Położenie reszty	Maksymalna wartość ilorazu
dzielnej	dzielnika					
16	8	AX	Dzielnik znajduje się w rejestrze lub w komórce pamięci podanej w polu operandu rozkazu DIV	AL	AH	255
32	16	DX:AX		AX	DX	65535
64	32	EDX:EAX		EAX	EDX	$2^{32}-1$
128	64	RDX:RAX		RAX	RDX	$2^{64}-1$

W powyższej tabelce występuje kilka nazw rejestrów, rozdzielonych znakiem dwukropka **:** . Przykładowo, EDX:EAX oznacza rejestr 64-bitowy złożony z dwóch rejestrów 32-bitowych EDX i EAX, przy czym w rejestrze EDX przechowywana jest starsza część liczby, a w rejestrze EAX – młodsza część liczby. Poniższy rysunek przedstawia sytuację, gdy w rejestrze EDX:EAX znajduje się dzielna.



Poniżej podano fragment programu w asemblerze, w którym następuje konwersja liczby binarnej bez znaku zawartej w rejestrze EAX na ciąg cyfr dziesiętnych tej liczby. Uzyskiwane cyfry są przekształcane na kod ASCII, a następnie wyświetlane na ekranie. W trakcie analizy podanego niżej kodu należy brać pod uwagę maksymalną zawartość rejestru EAX, która może wynosić $2^{32} - 1 = 4\ 294\ 967\ 295$. W systemie dziesiętnym liczba ta zajmuje 10 pozycji.

```
; obszar danych programu
.data
; deklaracja tablicy 12-bajtowej do przechowywania
; tworzonych cyfr
znaki               db   12 dup (?)
_   _   _   _   _   _   _   _   _   _   _   _
_   _   _   _   _   _   _   _   _   _   _   _
; obszar instrukcji (rozkazów) programu
.code

_   _   _   _   _   _   _   _   _   _   _   _

     mov       esi, 10        ; indeks w tablicy 'znaki'
     mov       ebx, 10        ; dzielnik równy 10

konwersja:
     mov       edx, 0    ; zerowanie starszej części dzielnej
     div       ebx       ; dzielenie przez 10, reszta w EDX,
                         ; iloraz w EAX
     add       dl, 30H   ; zamiana reszty z dzielenia na kod
                         ; ASCII
     mov       znaki [esi], dl; zapisanie cyfry w kodzie ASCII
     dec       esi            ; zmniejszenie indeksu
     cmp       eax, 0         ; sprawdzenie czy iloraz = 0
     jne       konwersja      ; skok, gdy iloraz niezerowy

; wypełnienie pozostałych bajtów spacjami i wpisanie
; znaków nowego wiersza
wypeln:
     or        esi, esi
     jz        wyswietl       ; skok, gdy ESI = 0
     mov       byte PTR znaki [esi], 20H ; kod spacji
     dec       esi            ; zmniejszenie indeksu
     jmp       wypeln

wyswietl:
     mov       byte PTR znaki [0], 0AH ; kod nowego wiersza
     mov       byte PTR znaki [11], 0AH ; kod nowego wiersza

; wyświetlenie cyfr na ekranie
     push      dword PTR 12   ; liczba wyświetlanych znaków
     push      dword PTR OFFSET znaki   ; adres wyśw. obszaru
     push      dword PTR 1; numer urządzenia (ekran ma numer 1)
     call      __write        ; wyświetlenie liczby na ekranie
     add       esp, 12        ; usunięcie parametrów ze stosu
```

Tworzenie i wywoływanie podprogramów

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.

Wywołanie ciągu rozkazów 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 sekwencji rozkazów tworzącej podprogram.

Wywołanie podprogramu realizuje się za pomocą rozszerzonego rozkazu skoku — konieczne jest bowiem zapamiętanie adresu powrotu, zwanego *śladem*, tj. miejsca, do którego ma powrócić sterowanie po zakończeniu wykonywania podprogramu. W architekturze *x86* ww. czynności wykonuje rozkaz `CALL`, który zapisuje adres powrotu na stosie.

Ślad zapisany na stosie wskazuje miejsce w programie, dokąd należy przekazać sterowanie po wykonaniu podprogramu. Innymi słowy: w chwili zakończenia wykonywania podprogramu zawartość wierzchołka stosu powinna zostać przepisana do rejestru EIP — czynności te realizuje rozkaz `RET`.

W asemblerze kod podprogramu rozpoczyna dyrektywa `PROC` a kończy dyrektywa `ENDP`, np.

```
czytaj          PROC
_   _   _   _   _   _
_   _   _   _   _   _
czytaj          ENDP
```

W celu wykonania podprogramu należy wprowadzić rozkaz `CALL`, np. `CALL czytaj`. W większości przypadków przed wywołaniem podprogramu trzeba także podać odpowiednie wartości parametrów podprogramu. Zazwyczaj parametry przekazywane są przez rejestry ogólnego przeznaczenia lub przez stos.

Na poprzednich stronach została opisana technika zamiany liczb binarnych na postać dziesiętną. Ponieważ w praktyce programowania na poziomie rozkazów procesora zamiana tak występuje dość często, warto przekształcić podany tam kod do postaci podprogramu, co pozwoli na łatwe wyświetlanie wyników programu, wszędzie gdzie jest to potrzebne.

Ponieważ przewidujemy, że przed wywołaniem podprogramu liczba binarna przeznaczona do wyświetlenia będzie znajdowała się w 32-bitowym rejestrze EAX, więc można przyjąć nazwę podprogramu `wyswietl_EAX`. W tej sytuacji pierwszy i ostatni wiersz podprogramu będą miały postać:

```
wyswietl_EAX   PROC
_   _   _   _   _   _
wyswietl_EAX   ENDP
```

Ze względu na wygodę programowania, jest pożądane aby w miarę możliwości podprogramy nie zmieniały zawartości rejestrów. Z tego względu na początku podprogramu zapamiętamy rejestry ogólnego przeznaczenia na stosie (rozkaz `PUSHA`), a w końcowej części podprogramu odtworzymy te rejestry (rozkaz `POPA`). Jak już wspomniano, powrót z podprogramu do programu głównego następuje wskutek wykonania rozkazu `RET`. Jeśli treść

podprogramu stanowić będą rozkazy podane na str. 2 i 3, to kod podprogramu będzie miał postać:

```
wyswietl_EAX   PROC
               pusha
– – – – – – – – – – – – – – –      (tu wpisać rozkazy
– – – – – – – – – – – – – – –       podane na stronach 2 i 3)
               popa
               ret
wyswietl_EAX   ENDP
```

Tworząc powyższy podprogram należy pamiętać o zarezerwowaniu w sekcji danych obszaru 12 bajtów, w którym zostaną przygotowane cyfry przeznaczone do wyświetlenia na ekranie:

```
.data
znaki              db   12 dup (?)
```

Zadanie 3.1. Napisać program w asemblerze, który wyświetli na ekranie 50 początkowych elementów ciągu liczb: 1, 2, 4, 7, 11, 16, 22, ... W programie wykorzystać podprogram `wyswietl_EAX`.

Wskazówka: struktura programu może być następująca:

```
.686
.model flat
extern    __write : PROC
extern    _ExitProcess@4 : PROC
public _main
.data
znaki              db   12 dup (?)
.code

wyswietl_EAX   PROC
               pusha
– – – – – – – – – – – – – – – – – – – –
– – – – – – – – – – – – – – – – – – – –
               popa
               ret
wyswietl_EAX   ENDP

_main PROC
– – – – – – – – – – – – – – – – – – – –
– – – – – – – – – – – – – – – – – – – –
               push 0
               call _ExitProcess@4
_main ENDP
END
```

Konwersja dziesiętno–dwójkowa

Zadanie konwersji dziesiętno-dwójkowej pojawia się, np. przy wczytywaniu liczb z klawiatury. Wczytywane są wtedy kody ASCII kolejnych cyfr wielocyfrowej liczby dziesiętnej. Przykładowo, naciśnięcie klawisza 7 powoduje, że do 8-bitowego rejestru procesora zostanie wprowadzony kod ASCII cyfry 7, który ma postać 00110111 (w zapisie szesnastkowym 37H). Analogicznie kodowane są inne cyfry, np. 6 ma kod 36H, 5 ma kod 35H, itd. Zatem zamiana kodu ASCII pojedynczej cyfry na jej wartość (tj. liczbę 0 – 9) polega po prostu na odjęciu od kodu ASCII wartości 30H.

Właściwą konwersję wykonujemy na zawartościach rejestrów, w których naturalną reprezentacją jest postać dwójkowa. Sposób konwersji może być następujący. Wartość pewnej liczby dziesiętnej zapisanej za pomocą cyfr $x_n, x_{n-1},,, x_2, x_1, x_0$ można zapisać następująco:

$$x_n \cdot 10^n + x_{n-1} \cdot 10^{n-1} + \ldots + x_2 \cdot 10^2 + x_1 \cdot 10^1 + x_0 \cdot 10^0$$

W praktyce programowania wygodniej jednak posługiwać się schematem iteracyjnym:

$$(...(((x_n \cdot 10 + x_{n-1}) \cdot 10 + x_{n-2}) \cdot 10 + x_{n-3}) \cdot 10 + ...) \cdot 10 + x_0$$

Przykładowo, jeśli użytkownik wprowadza z klawiatury liczbę 5804, czyli wprowadza kolejno cyfry 5, 8, 0, 4, to wartość wprowadzanej liczby wynosi:

$$((5 \cdot 10 + 8) \cdot 10 + 0) \cdot 10 + 4$$

W ten właśnie sposób, dysponując cyframi dziesiętnymi liczby możemy obliczyć jej wartość.

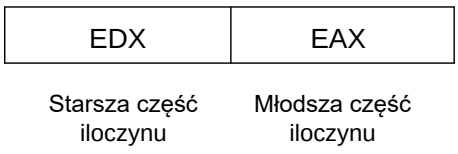
Kodowanie algorytmu będzie nieco łatwiejsze, jeśli przyjąć że użytkownik wcześniej wprowadził już cyfrę 0 (co oczywiście nie wpływa na wynik końcowy). W tej sytuacji, po wprowadzeniu przez użytkownika cyfry 5, mnożymy wcześniej uzyskany wynik przez 10 i dodajemy 5. Jeśli użytkownik wprowadzi cyfrę 8, to tak jak poprzednio, mnożymy dotychczas uzyskany wynik przez 10 i dodajemy 8. Tak samo postępujemy przy kolejnych cyfrach.

$$(((0 \cdot 10 + 5) \cdot 10 + 8) \cdot 10 + 0) \cdot 10 + 4$$

Zauważmy, że w podanym algorytmie nie określa się z góry ilości cyfr — wymaga się jedynie aby reprezentacja binarna wprowadzonej liczby dała się zapisać na 32 bitach.

Poniżej podano fragment programu, w którym przeprowadzana jest omawiana konwersja. Dodatkowo zakładamy, że wartość wprowadzanej liczby nie przekracza $2^{32} - 1$ (tzn. postać binarna liczby da się przedstawić na 32 bitach), przy czym naciśnięcie klawisza Enter (kod 10) traktowane jest jako zakończenie wprowadzania cyfr liczby.

W omawianym fragmencie mnożenie przez 10 wykonywane jest za pomocą rozkazu `mul esi`. Rozkaz ten wykonuje mnożenie liczb bez znaku: mnożna zawarta jest w rejestrze EAX, mnożnik (w tym przykładzie) w rejestrze ESI. Wynik mnożenia jest liczbą 64-bitową, która wpisywana jest do rejestru EDX:EAX, tj. starsza część iloczynu wpisywana jest do rejestru EDX, a młodsza część iloczynu do rejestru EAX.



Ponieważ założyliśmy, że wartość wprowadzanej liczby nie przekracza $2^{32} - 1$, więc starsza część iloczynu (w rejestrze EDX) będzie równa 0 i może być pominięta w dalszych obliczeniach.

```
; wczytywanie liczby dziesiętnej z klawiatury – po
; wprowadzeniu cyfr należy nacisnąć klawisz Enter

; liczba po konwersji na postać binarną zostaje wpisana
; do rejestru EAX

; deklaracja tablicy do przechowywania wprowadzanych cyfr
; (w obszarze danych)
obszar    db   12 dup (?)
dziesiec       dd   10   ; mnożnik
_   _   _   _   _   _   _   _   _   _   _   _

; max ilość znaków wczytywanej liczby
     push      dword PTR 12

     push      dword PTR OFFSET obszar  ; adres obszaru pamięci
     push      dword PTR 0; numer urządzenia (0 dla klawiatury)
     call      __read    ; odczytywanie znaków z klawiatury
                    ; (dwa znaki podkreślenia przed read)
     add       esp, 12   ; usunięcie parametrów ze stosu


; bieżąca wartość przekształcanej liczby przechowywana jest
; w rejestrze EAX; przyjmujemy 0 jako wartość początkową
     mov            eax, 0
     mov       ebx, OFFSET obszar  ; adres obszaru ze znakami

pobieraj_znaki:
     mov       cl, [ebx] ; pobranie kolejnej cyfry w kodzie
                         ; ASCII
     inc       ebx       ; zwiększenie indeksu
     cmp       cl,10     ; sprawdzenie czy naciśnięto Enter
     je        byl_enter ; skok, gdy naciśnięto Enter
     sub       cl, 30H   ; zamiana kodu ASCII na wartość cyfry
     movzx     ecx, cl   ; przechowanie wartości cyfry w
                         ; rejestrze ECX

     ; mnożenie wcześniej obliczonej wartości razy 10
     mul       dword PTR dziesiec
     add       eax, ecx  ; dodanie ostatnio odczytanej cyfry
     jmp       pobieraj_znaki ; skok na początek pętli

byl_enter:
; wartość binarna wprowadzonej liczby znajduje się teraz w
rejestrze EAX
```

Zadanie 3.2. Przekształcić powyższy fragment programu do postaci podprogramu o nazwie `wczytaj_do_EAX`.
Wskazówka: rozkazy `PUSHA` i `POPA` nie nadają się do tego podprogramu (dlaczego?) — zamiast tych rozkazów, na początku podprogramu trzeba zapisać na stosie rejestry ogólnego przeznaczenia używane w podprogramie, a w końcowej części podprogramu odtworzyć te rejestry. Nie trzeba zapisywać zawartości rejestrów EAX i ESP.

Zadanie 3.3. Napisać program, który wczyta z klawiatury liczbę dziesiętną mniejszą od 60000 i wyświetli na ekranie kwadrat tej liczby. W programie wykorzystać podprogramy `wczytaj_do_EAX` i `wyswietl_EAX`.

Konwersja dwójkowo–szesnastkowa

Konwersja liczby z postaci binarnej na szesnastkową jest stosunkowo prosta: wystarczy tylko przyporządkować kolejnym grupom czterech bitów w liczbie binarnej odpowiednią cyfrę ze zbioru 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Przykładowo, grupie 1011 przyporządkujemy cyfrę B. Poniżej podany jest kod podprogramu, który przeprowadza taką konwersję.

Istotną rolę w trakcie konwersji pełni 16-bajtowa tablica, w której umieszczone są kody ASCII wszystkich cyfr używanych w systemie szesnastkowym. Postać tej tablicy w kodzie asemblerowym jest następująca:

```
.data
dekoder   db   '0123456789ABCDEF'
```

W trakcie konwersji, z liczby binarnej wydzielamy kolejno grupy 4-bitowe i traktujemy każdą czwórkę bitów jako liczbę binarną, która może przyjmować wartości z przedziału 0 ÷ 15. Liczba ta jest następnie traktowana jako indeks w tablicy `dekoder`, wskazując jeden z elementów tej tablicy. Przykładowo, jeśli czwórka bitów ma postać 0110 (dziesiętnie 6), to z tablicy `dekoder`, za pomocą podanego niżej rozkazu

```
mov       dl, dekoder[ebx] ; pobranie cyfry z tablicy
```

zostaje odczytany szósty element tablicy, którym jest kod ASCII cyfry '6'. W rezultacie kod ASCII cyfry '6' zostanie wpisany do rejestru `dl`. Przed wykonaniem powyższego rozkazu na cztery ostatnie bity rejestru `ebx` powinna zostać wpisana omawiana czwórka bitów, przy czym pozostałe bity rejestru `ebx` powinny być wyzerowane.

W trakcie wykonywania powyższego rozkazu procesor wyznaczy adres komórki pamięci zawierającej potrzebny bajt jako sumę adresu początkowego tablicy `dekoder` i zawartości rejestru `ebx`. Następnie procesor odczyta bajt z komórki pamięci o obliczonym adresie i załaduje go do rejestru `dl`.

Warto zwrócić uwagę na rezerwację i wykorzystanie obszaru roboczego na stosie. Podobnie jak w przypadku podprogramu konwersji z postaci binarnej na dziesiętną, tak i tutaj tworzone cyfry w zapisie szesnastkowym muszą być tymczasowo przechowywane w obszarze roboczym. W tym celu w poprzednim podprogramie w sekcji danych zarezerwowano 12-bajtowy obszar `znaki`. Obszar ten potrzebny jest jednak tylko w czasie wykonywania kodu podprogramu, a umieszczenie go w sekcji danych (`.data`) niepotrzebnie

zajmuje pamięć przez cały czas wykonywania programu. Lepszym rozwiązaniem jest zarezerwowanie na stosie obszaru 12-bajtów i zwolnienie go w końcowej części podprogramu. W podanym niżej podprogramie rezerwacja 12 bajtów na stosie realizowana jest za pomocą rozkazu `sub esp, 12`, a zwolnienie obszaru za pomocą rozkazu `add esp, 12`. Bezpośrednio po przydzieleniu obszaru jego adres wpisywany jest do rejestru EDI.

```
wyswietl_EAX_hex    PROC
; wyświetlanie zawartości rejestru EAX
; w postaci liczby szesnastkowej

     pusha          ; przechowanie rejestrów

; rezerwacja 12 bajtów na stosie (poprzez zmniejszenie
; rejestru ESP) przeznaczonych na tymczasowe przechowanie
; cyfr szesnastkowych wyświetlanej liczby
     sub       esp, 12
     mov       edi, esp  ; adres zarezerwowanego obszaru
                         ; pamięci

; przygotowanie konwersji
     mov       ecx, 8    ; liczba obiegów pętli konwersji
     mov       esi, 1    ; indeks początkowy używany przy
                         ; zapisie cyfr

; pętla konwersji
ptl3hex:

; przesunięcie cykliczne (obrót) rejestru EAX o 4 bity w lewo
; w szczególności, w pierwszym obiegu pętli bity nr 31 - 28
; rejestru EAX zostaną przesunięte na pozycje 3 - 0
     rol       eax, 4

; wyodrębnienie 4 najmłodszych bitów i odczytanie z tablicy
; 'dekoder' odpowiadającej im cyfry w zapisie szesnastkowym
     mov       ebx, eax  ; kopiowanie EAX do EBX
     and       ebx, 0000000FH ; zerowanie bitów 31 - 4  rej.EBX
     mov       dl, dekoder[ebx] ; pobranie cyfry z tablicy

; przesłanie cyfry do obszaru roboczego
     mov       [edi][esi], dl

     inc       esi       ;inkrementacja modyfikatora
     loop      ptl3hex   ; sterowanie pętlą

; wpisanie znaku nowego wiersza przed i po cyfrach
     mov       byte PTR [edi][0], 10
     mov       byte PTR [edi][9], 10
```

```
; wyświetlenie przygotowanych cyfr
     push      10   ; 8 cyfr + 2 znaki nowego wiersza
     push      edi  ; adres obszaru roboczego
     push      1    ; nr urządzenia (tu: ekran)
     call      __write   ; wyświetlenie



; usunięcie ze stosu 24 bajtów, w tym 12 bajtów zapisanych
; przez 3 rozkazy push przed rozkazem call
; i 12 bajtów zarezerwowanych na początku podprogramu
     add       esp, 24

     popa      ; odtworzenie rejestrów
     ret       ; powrót z podprogramu
wyswietl_EAX_hex    ENDP
```

Zadanie 3.4. Napisać program w asemblerze, który wczyta liczbę dziesiętną z klawiatury i wyświetli na ekranie jej reprezentację w systemie szesnastkowym. W programie wykorzystać podprogramy `wczytaj_do_EAX` i `wyswietl_EAX_hex`.

Zadanie 3.5. Zmodyfikować podprogram `wyswietl_EAX_hex` w taki sposób, by w wyświetlanej liczbie szesnastkowej zera nieznaczące z lewej strony zostały zastąpione spacjami.

Konwersja szesnastkowo–dwójkowa

Konwersja liczb z postaci szesnastkowej na binarną dotyczy głównie sytuacji, gdy liczba w zapisie szesnastkowym wprowadzana jest z klawiatury. Podobnie jak w przypadku, gdy wykonywana jest konwersja liczby dziesiętnej, każdy znak ze zbioru 0, 1, 2, ..., F przekształca się na odpowiadający mu 4-bitowy kod binarny. Kody te umieszcza się na kolejnych czwórkach bitach w rejestrze wynikowym, tworząc w ten sposób 32-bitową liczbę binarną. Poniżej podano przykład podprogramu, który wykonuje taką konwersję. Podprogram akceptuje cyfry ze zbioru: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, e, E, f, F.

W omawianym podprogramie zamiana cyfr w kodzie ASCII 0, 1, 2, ..., 9 (wartości 30H, 31H, 32H, ..., 39H) na odpowiedni kod binarny realizowana jest przez proste odejmowanie: od kodu ASCII cyfry odejmowany jest kod cyfry 0 (30H) — w rezultacie uzyskuje się 4-bitowe wartości binarne odpowiadające poszczególnym cyfrom: 0000, 0001, 0010, ..., 1001. Odejmowanie wykonuje rozkaz `sub dl, '0'`.

Zamiana ta jest nieco bardziej skomplikowana w odniesieniu do cyfr z podzbioru a, A, b, B, c, C, e, E, f, F. Zauważmy, że cyfra `a` lub `A` powinna być zamieniona na kod 1010, cyfra `b` lub `B` na kod 1011, ..., cyfra `f` lub `F` na kod 1111. Biorąc pod uwagę, że kod ASCII cyfry `A` wynosi 41H, kod cyfry `B` wynosi 42H, itd. w celu uzyskania kodu 4-bitowego należy najpierw odjąć 41H (kod cyfry `A`), a następnie dodać 10, czyli konwersja dla cyfr z podzbioru A, B, C, E, F przeprowadzana jest wg formuły:

kod wynikowy = kod ASCII cyfry – 41H + 10 = kod ASCII cyfry – (41H – 10)

W podanym niżej podprogramie działanie to wykonuje rozkaz `sub dl, 'A' - 10`. Analogicznie przeprowadza się konwersję dla cyfr z podzbioru `a, b, c, d, e, f` — w tym przypadku rozkaz ma postać `sub dl, 'a' - 10`.

```
wczytaj_do_EAX_hex  PROC

; wczytywanie liczby szesnastkowej z klawiatury - liczba po
; konwersji na postać binarną zostaje wpisana do rejestru EAX
; po wprowadzeniu ostatniej cyfry należy nacisnąć klawisz
; Enter

     push      ebx
     push      ecx
     push      edx
     push      esi
     push      edi
     push      ebp

; rezerwacja 12 bajtów na stosie przeznaczonych na tymczasowe
; przechowanie cyfr szesnastkowych wyświetlanej liczby
     sub       esp, 12   ; rezerwacja poprzez zmniejszenie ESP
     mov       esi, esp ; adres zarezerwowanego obszaru pamięci

     push      dword PTR 10 ; max ilość znaków wczytyw. liczby
     push      esi       ; adres obszaru pamięci
     push      dword PTR 0; numer urządzenia (0 dla klawiatury)
     call      __read    ; odczytywanie znaków z klawiatury
                         ; (dwa znaki podkreślenia przed read)
     add       esp, 12   ; usunięcie parametrów ze stosu

     mov       eax, 0    ; dotychczas uzyskany wynik

pocz_konw:
     mov       dl, [esi] ; pobranie kolejnego bajtu
     inc       esi       ; inkrementacja indeksu
     cmp       dl, 10    ; sprawdzenie czy naciśnięto Enter
     je        gotowe    ; skok do końca podprogramu

; sprawdzenie czy wprowadzony znak jest cyfrą 0, 1, 2 , ..., 9
     cmp       dl, '0'
     jb        pocz_konw ; inny znak jest ignorowany
     cmp       dl, '9'
     ja        sprawdzaj_dalej
     sub       dl, '0'   ; zamiana kodu ASCII na wartość cyfry
dopisz:
     shl       eax, 4 ; przesunięcie logiczne w lewo o 4 bity
     or        al, dl ; dopisanie utworzonego kodu 4-bitowego
```

```
                    ; na 4 ostatnie bity rejestru EAX
     jmp       pocz_konw ; skok na początek pętli konwersji

; sprawdzenie czy wprowadzony znak jest cyfrą A, B, ..., F
sprawdzaj_dalej:
     cmp       dl, 'A'
     jb        pocz_konw      ; inny znak jest ignorowany
     cmp       dl, 'F'
     ja        sprawdzaj_dalej2
     sub       dl, 'A' - 10   ; wyznaczenie kodu binarnego
     jmp       dopisz

; sprawdzenie czy wprowadzony znak jest cyfrą a, b, ..., f
sprawdzaj_dalej2:
     cmp       dl, 'a'
     jb        pocz_konw   ; inny znak jest ignorowany
     cmp       dl, 'f'
     ja        pocz_konw   ; inny znak jest ignorowany
     sub       dl, 'a' - 10
     jmp       dopisz

gotowe:
; zwolnienie zarezerwowanego obszaru pamięci
     add       esp, 12

     pop       ebp
     pop       edi
     pop       esi
     pop       edx
     pop       ecx
     pop       ebx
     ret

wczytaj_do_EAX_hex  ENDP
```

Zadanie 3.6. Napisać program w asemblerze, który wczyta liczbę szesnastkową z klawiatury i wyświetli na ekranie jej reprezentację w systemie dziesiętnym. W programie wykorzystać podprogramy `wczytaj_do_EAX_hex` i `wyswietl_EAX`.