Pytania i zadania przykładowe do ćwiczeń z „Architektury komputerów”

cz. I, listopad 2015

---

1. W czterech kolejnych bajtach pamięci znajdują się liczby:

adres	*zawartość*
65BH	2AH
65AH	00H
659H	37H
658H	F2H

Przyjmując, że podane bajty stanowią liczbę binarną 32-bitową podać wartość tej liczby w zapisie szesnastkowym przy założeniu, że stosowana jest konwencja *mniejsze wyżej* (ang. big endian).

2. Przed wykonaniem poniższego fragmentu programu w rejestrze EAX znajdowała się liczba p, w rejestrze EDX – liczba q. Określić zawartość tych rejestrów po wykonaniu poniższych rozkazów.

```
xor   eax, edx
xor   edx, eax
xor   eax, edx
```

Wskazówki: określić zawartość k-tego bitu rejestru EDX, jeśli wiadomo, że przed wykonaniem podanego fragmentu programu na bitach o numerze k znajdowały się wartości a_k (rejestr EAX) i d_k (rejestr EDX). Ponadto operacja XOR:

- jest przemienna $x \oplus y = y \oplus x$
- jest łączna $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- zachodzi związek $x \oplus x = 0$.

3. Napisać fragment programu w asemblerze, który wykona działania równoważne działaniu poniższego rozkazu

```
xor   edi, esi
```

W napisanym fragmencie nie można używać rozkazu `xor`.

4. Wyjaśnić dlaczego wykonanie poniższego fragmentu programu spowoduje wygenerowanie wyjątku procesora?

```
mov   ax, 0
mov   dx, 1
div   dx
```

5. Na czym polega różnica w sposobie wykonania poniższych rozkazów:

```
push  dword PTR esi
push  dword PTR [esi]
```

6. Podać zawartości rejestrów EBX i CX po wykonaniu niżej podanego fragmentu programu

```
.data
stale       DW 2,1
napis       DW 10 dup (3),2
tekst       DB 7
            DQ 1
.code
_main:
      MOV    CX, napis -1
      SUB    tekst, CH
      MOV    EDI,1
      MOV    tekst[4*EDI],CH
      MOV  EBX, DWORD PTR tekst+1
```

7. Poniższy fragment programu może służyć do rezerwacji obszaru pamięci na dane o nieokreślonych wartościach początkowych. Podać równoważną deklarację tego obszaru używając dyrektywy `dd`.

```
obroty      LABEL       dword
            ORG         $ + 28
```

8. Określić zawartości znaczników OF, ZF i CF po wykonaniu podanego niżej fragmentu programu.

```
      xor   eax, eax
      sub   eax, 0FFFFFFFFH
```

9. W wyniku wykonania podanego niżej fragmentu programu w języku C, zmiennej p została przypisana wartość 0x12. Określić czy w komputerze stosowana jest konwencja mniejsze niżej (little endian) czy mniejsze wyżej (big endian).

```
unsigned char p ;
unsigned short int proba = 0x1234 ;
unsigned char * wsk =
    (unsigned char *) &proba ;
p = *wsk ;
```

10. W rejestrze EBX znajduje się liczba całkowita w kodzie U2. Zakładamy, że liczba zawarta jest w przedziale $< -(2^{31} - 1), 2^{31} - 1>$. Napisać fragment, który przekoduje tę liczbę na kod *znak-moduł*.

11. Podać zawartość rejestru `DH` w postaci liczby dziesiętnej po wykonaniu poniższych rozkazów:

```
      mov   dh, 15
      xor   dh, 12
```

12. Uzupełnić zdanie: *W wyniku wykonania poniższego rozkazu zawartość rejestru ESI zostanie . . . . . .*

```
lea   esi, [esi + esi*8]
```

13. Na czym polega błąd w poniższym fragmencie programu:

```
sub   esp, 4
mov   [esp], 'A'
```

14. Rozkazy

```
push  ebx
push  ecx
```

można zastąpić równoważna sekwencją:

```
sub    esp, 8
mov   [...], ebx
mov   [...], ecx
```

Uzupełnić pola adresowe podanych wyżej rozkazów `mov`.

15. Jakie wartości zostaną wpisane do rejestrów EDX i EAX po wykonaniu niżej podanego fragmentu programu?

```
mov   eax, 0FFFFFFFFH
mov   ebx, 0FFFFFFFFH
imul  ebx
```

16. Na czym polega błąd w poniższym fragmencie programu?

```
v2    dw    ?
-  -  -  -  -  -  -  -  -  -  -  -
      mov   v2, 11111H
```

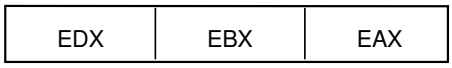
17. Określić zawartości znaczników OF, ZF i CF po wykonaniu podanego niżej fragmentu programu.

```
mov   ax, 1
add   ax, 0FFFFH
```

18. Podać liczbę, która zostanie wyświetlona na ekranie w wyniku wykonania poniższego fragmentu programu. Podprogram wyswietl32 wyświetla na ekranie w postaci dziesiętnej liczbę binarną zawartą w rejestrze EAX.

```
qxy   dw    254, 255, 256
_     _     _     _     _     _     _     _
      mov   eax, dword PTR qxy + 1
      call  wyswietl32
```

19. W rejestrach EDX:EBX:EAX znajduje się 96-bitowy ciąg bitów. Napisać fragment programu, w którym ciąg ten zostanie przesunięty cyklicznie w lewo o 1 pozycję.



Wskazówka: wykorzystać rozkazy przesunięcia w lewo SHL (bity wychodzące z rejestru wpisywane są do CF) i RCL (zawartość CF wpisywana jest na najmłodszy bit rejestru, a bity wychodzące z rejestru wpisywane są do CF). Wykorzystać także rozkaz BT.

20. Napisać fragment programu w asemblerze, który zamieni młodszą (bity 15 – 0) i starszą (bity 31 – 16) część rejestru EDX.

21. W czterech kolejnych bajtach pamięci począwszy od adresu podanego w rejestrze w EBX znajduje się 32-bitowa liczba całkowita bez znaku zakodowana w formacie *mniejsze wyżej* (big endian). Nie używając rozkazu BSWAP załadować tę liczbę do rejestru EAX w formacie *mniejsze niżej* (little endian).

22. Napisać fragment programu w asemblerze, który obliczy liczbę bitów o wartości 1 zawartych w rejestrze EAX. Wynik obliczenia wpisać do rejestru CL.

23. Napisać fragment programu, w którym liczba 32-bitowa bez znaku znajdująca się w rejestrze EAX zostanie pomnożona przez 10 (dziesięć). Wynik mnożenia w postaci liczby 32-bitowej powinien zostać wpisany do rejestru EAX. Zakładamy, że mnożenie nie doprowadzi do powstania nadmiaru. W omawianym fragmencie nie mogą być używane rozkazy MUL lub IMUL. *Wskazówka*: wykorzystać rozkazy przesunięć i zależność $a \cdot 10 = a \cdot 8 + a \cdot 2$.

24. Ile bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze?

```
v1    dq    ?, ?
v2    dw    4 dup (?), 20
v3    db    10 dup (?)
```

25. Na czym polega błąd w poniższym fragmencie programu?

```
const2      db    ?
-  -  -  -  -  -  -  -  -  -  -
      mov   const2, 256
```

26. Wyjaśnić działanie poniższego fragmentu programu

```
start:      mov   ecx, 3
            sub   ax, 10
            loop  start
```

27. Podać zawartość rejestru EIP po wykonaniu poniższej sekwencji rozkazów

```
      mov   edx, 347
      xchg  [esp], edx
      ret
```

28. Dla podanych niżej dwóch rozkazów podać równoważny ciąg rozkazów, w którym nie wystąpi rozkaz `loop`.

```
            loop  oblicz
   oblicz:  add   dh, 7
```

29. Na czym polega błąd w podanym niżej zapisie rozkazu

```
mov byte PTR [eax], byte PTR [edx]
```

30. W pewnym programie została zdefiniowana zmienna

```
   wskaznik       dd     ?
```

Napisać fragment programu w asemblerze, który wpisze do tej zmiennej adres komórki pamięci, w której znajduje się ta zmienna.

31. Jaka wartość zostanie wprowadzona do rejestru EDX po wykonaniu podanego niżej fragmentu programu

```
linie dd    421, 422, 443,
      dd    442, 444, 427, 432
_  _  _  _  _  _  _  _  _  _
      mov   esi, (OFFSET  linie)+4
      mov   ebx, 4
      mov   edx, [ebx] [esi]
```

32. Napisać fragment programu w asemblerze, który obliczy sumę cyfr dziesiętnych liczby zawartej w rejestrze EAX. Wynik obliczenia wpisać do rejestru CL. Przykład: jeśli w rejestrze EAX znajduje się liczba 1111101 (dziesiętnie 125), to po wykonaniu fragmentu rejestr CL powinien zawierać 00001000.

33. Określić postać komunikatu wyświetlanego przez funkcję `MessageBox` po wykonaniu poniższego fragmentu programu.

```
napis db 'informatyka', 0, 4 dup (?)
_    _    _    _    _    _    _    _
            mov   ecx, 12
przepisz:   mov   al, napis[ecx-1]
            mov   napis[ecx+3], al
            loop  przepisz

            push  0
            push  OFFSET napis
            lea   eax, napis[3]
            push  eax
            push  0
            call  _MessageBoxA@16
```

34. W tablicy znaki znajduje się pewien tekst zakodowany w formacie UTF–8. Tekst zakończony jest bajtem o wartości 0. Napisać fragment programu w asemblerze, który wyznaczy liczbę bajtów, które zajmować będzie ww. tekst po zamianie na 16-bitowy format UTF-16. Obliczoną liczbę bajtów wpisać do rejestru ECX. Przyjąć, że tekst w zawiera znaki zakodowane na jednym, dwóch lub trzech bajtach. Reguły kodowania opisuje poniższa tabela:

Zakresy od ...do..		Kodowanie UTF–8
00H	7FH	0*xxxxxxx*
80H	7FFH	110*xxxxx* 10*xxxxxx*
800H	FFFFH	1110*xxxx* 10*xxxxxx* 10*xxxxxx*

35. Podany poniżej podprogram `dodaj` sumuje dwie liczby 32-bitowe umieszczone bezpośrednio za rozkazem `call`, który wywołuje ten podprogram. Obliczona suma pozostawiana jest w rejestrze EAX.

```
dodaj PROC
      mov   esi, [esp]
      mov   eax, [esi]
      add   eax, [esi+4]
      ret
dodaj ENDP
```

Przykładowe wywołanie podprogramu może mieć postać:

```
       call  dodaj
       dd    5
       dd    7
       jmp   ciag_dalszy
```

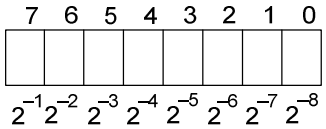
Wyjaśnić dlaczego wywołanie podanego podprogramu może spowodować bliżej nieokreślone działania procesora, prowadzące do błędu wykonania programu? Następnie, do podanego kodu podprogramu wprowadzić dodatkowe rozkazy, które wyeliminują ww. błędne działania.

36. W programach obsługi kalendarza MS Visual Studio dni świąteczne w miesiącu koduje się w postaci jedynek umieszczonych na odpowiednich bitach słowa 32-bitowego. Tablica zawierająca 12 takich elementów pozwala zakodować informacje obejmujące rok.
Napisać podprogram w asemblerze wyświetlający na ekranie daty dni świątecznych w podanym miesiącu. Parametry wywołania podprogramu znajdują się w rejestrach:
CL – numer miesiąca (1 – 12)
EBX – adres tablicy zawierającej zakodowane daty dni świątecznych w poszczególnych miesiącach.

37. W obszarze pamięci, którego adres początkowy zawarty jest w rejestrze ESI znajduje się ciąg cyfr (w kodzie ASCII) zakończony bajtem o wartości 0. Podany ciąg cyfr należy przepisać do obszaru pamięci o adresie podanym w rejestrze EDI, dodatkowo wprowadzając znak kropki przed dwie ostatnie cyfry. Jeśli w obszarze źródłowym znajdują się tylko dwie cyfry, to przed kropką należy wprowadzić dodatkową cyfrę 0 (przykład: 0.53). Jeśli zaś w obszarze źródłowym znajduje się tylko jedna cyfra, to dodatkowe zera trzeba wprowadzić przed i po kropce (przykład: 0.07).

38. W rejestrze EBX znajduje skompresowany ciąg 16 cyfr dziesiętnych. Kolejne pary bitów tego rejestru (tj. bity 30-31, 28-29, 26-27, itd.) zawierają kody przyporządkowane cyfrom kodu ASCII wg schematu: 00 → '3', 01 → '4', 10 → '7', 11 → '9'. Napisać fragment programu w asemblerze, który wykona dekompresję zawartości rejestru EBX, przy czym uzyskane cyfry (w kodzie ASCII) należy wpisać do 16-bajtowej tablicy iskry w sekcji danych.

39. W pewnym programie przyjęto reprezentację ułamków właściwych w postaci liczb 8-bitowych binarnych.



Napisać podprogram w asemblerze, który wyświetli na ekranie w postaci dziesiętnej zawartość rejestru AL z dokładnością 3 cyfr po kropce.. Liczba w rejestrze AL zakodowana jest w podanym wyżej formacie.
Przykład: jeśli w rejestrze AL znajduje się liczba binarna 11000000, to na ekranie powinna pojawić liczba 0.750
Konwersję na postać dziesiętną można przeprowadzić w poniższy sposób:
a. Liczbę w rejestrze AL mnożymy przez 10 używając rozkazu mnożenia dwóch liczb 8-bitowych – wynik mnożenia zostaje wpisany do rejestru AX.
b. Liczba wpisana do rejestru AH stanowi wartość kolejnej cyfry liczby dziesiętnej, zaczynając od najstarszej.
c. Powtarzamy opisane operacje wymaganą ilość razy.
d. Zamieniamy uzyskane wartości na kody ASCII i wyświetlamy na ekranie za pomocą funkcji *write*.

40. Napisać podprogram w asemblerze, który wyświetli na ekranie zawartość rejestru AL w postaci liczby dziesiętnej, wykorzystując niżej opisany algorytm. Zakładamy, że w rejestrze AL znajduje się 8-bitowa liczba binarna bez znaku. Konwersja (używana zazwyczaj do kodu BCD) przebiega następująco:
a. Wyzerować pozostałe bity rejestry EAX (z wyjątkiem AL).
b. Zbadać czy liczba umieszczona na bitach 3 – 0 rejestru AH jest większa od 4, jeśli tak, to do rejestru EAX dodać liczbę 300H.
c. Zbadać czy liczba umieszczona na bitach 7 – 4 rejestru AH jest większa od 4, jeśli tak, to do rejestru EAX dodać liczbę 3000H.
d. Przesunąć rejestr EAX o 1 pozycję w lewo.
e. Powtórzyć 8 razy czynności opisane w pkt. b., c., d.
f. Po przeprowadzeniu ww. operacji w rejestrze EAX znajdować się będą wartości pozycji: setek na bitach 19 – 16, dziesiątek na bitach 15 – 12, jedności na bitach 11 – 8. Wartości te zamienić na kod ASCII i wyświetlić na ekranie za pomocą funkcji *write*.

41. Dwie liczby całkowite dziesiętne bez znaku, zakodowane w postaci ciągu cyfr w kodzie ASCII, zostały umieszczone w pamięci głównej (operacyjnej). Każdy ciąg cyfr zakończony jest bajtem o wartości 0, a położenie obu ciągów w pamięci określone jest przez zawartości rejestrów ESI i EDI. Napisać fragment programu w asemblerze, który porówna obie liczby i ustawi znaczniki ZF i CF w niżej podany sposób.

{ESI} > {EDI}	⇒	CF = 0 i ZF = 0
{ESI} = {EDI}	⇒	CF = 0 i ZF = 1
{ESI} < {EDI}	⇒	CF = 1 i ZF = 0

Uwagi:

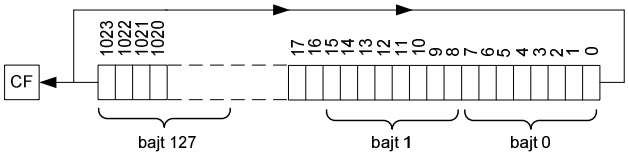
a. Zapisy {ESI} i {EDI} oznaczają, odpowiednio, wartości liczb wskazywanych przez rejestry ESI i EDI.
b. Liczby mogą mieć niejednakową liczbę cyfr.
c. Operację porównania przeprowadzić bez konwersji obu liczb na postać binarną.

42. Dwa wyrazy, złożone wyłącznie z małych liter alfabetu łacińskiego, zakodowane w kodzie ASCII w postaci dwóch ciągów bajtów, zostały umieszczone w pamięci głównej (operacyjnej). Każdy ciąg bajtów zakończony jest bajtem o wartości 0, a położenie obu ciągów w pamięci określone jest przez zawartości rejestrów ESI i EDI. Napisać fragment programu w asemblerze, który porówna oba wyrazy i ustawi znaczniki ZF i CF w niżej podany sposób.

CF = 0 i ZF = 0 jeśli wyraz wskazany przez rejestr ESI powinien być umieszczony w słowniku przed wyrazem wskazanym przez rejestr EDI;
CF = 1 i ZF = 0 jeśli wyraz wskazany przez rejestr ESI powinien być umieszczony w słowniku po wyrazie wskazanym przez rejestr EDI;
CF = 0 i ZF = 1 jeśli wyrazy są identyczne.

Wskazówki:
a. Wyrazy mogą mieć niejednakową liczbę liter.
b. Znacznik CF można ustawić za pomocą rozkazów STC (CF ← 1) lub CLC (CF ← 0).

43. W pewnym programie używany jest rejestr 1024-bitowy, który symulowany jest za pomocą tablicy *rejestr1024*, zawierającej 128 bajtów.



Napisać podprogram w asemblerze, który przesunie zawartość tego rejestru cyklicznie o 1 pozycję w lewo, przy czym bit wychodzący zostanie także wpisany do znacznika CF (podobnie jak w rozkazie ROL).

Wskazówki:
a. Przyjąć, że tablica *rejestr1024* została wcześniej zdefiniowana w sekcji danych programu jako tablica 128-bajtowa.
b. Wykorzystać m.in. rozkazy SHL i RCL.
c. W trakcie przenoszenia bitów z bajtu do bajtu za pomocą znacznika CF należy pamiętać, że znaczna liczba rozkazów wpływa na stan tego znacznika, natomiast m.in. rozkazy MOV, LOOP, INC, DEC nie wpływają na stan CF.

44. Napisać podprogram w asemblerze, który wyznaczy iloraz dwóch liczb całkowitych bez znaku wg podanego niżej algorytmu. Zakładamy, przez wywołaniem podprogramu dzielna (32-bitowa) znajduje się w rejestrze EAX, a dzielnik (16-bitowy) znajduje się w rejestrze BX. Iloraz z dzielenia należy wpisać do rejestru AX, a resztę z dzielenia do rejestru DX. W podprogramie nie można używać rozkazów DIV i IDIV. Zakładamy, że dla podanych wartości dzielnej i dzielnika, dzielenie nie doprowadzi do powstania nadmiaru.
Wskazówki:
a. Opisany algorytm dzielenia polega na pobieraniu kolejnych bitów dzielnej i porównywaniu pobranych wartości z dzielnikiem,
b. W zadaniu nie ma wymagań dotyczących przechowywania zawartości rejestrów.
c. Dla wygody dalszych obliczeń wartość dzielnika w rejestrze BX należy rozszerzyć do rejestru EBX (wartość dzielnika nie przekracza 65535). Iloraz z dzielenia będzie tworzony tymczasowo w rejestrze EDI (inicjalnie wyzerowanym). Rejestr EDX (inicjalnie wyzerowany) będzie służył jako rejestr roboczy. Opisane niżej działania należy powtórzyć 32 razy.
d. Przesunąć rejestr EAX o 1 pozycję w lewo i wpisać wysunięty bit na najmłodszą pozycję rejestru EDX (uprzednio przesuwając go o 1 pozycję w lewo).
e. Jeśli liczba w EDX jest większa lub równa od dzielnika, to rejestr EDI należy przesunąć o 1 pozycję w lewo i wpisać jedynkę na najmłodszy bit. Ponadto od liczby w rejestrze EDX należy odjąć dzielnik.
f. Jeśli liczba w EDX jest mniejsza od dzielnika, to rejestr EDI należy przesunąć o 1 pozycję w lewo.

45. Przyjmując, że wartość punktu kodowego Unikodu należy do przedziału <10000H, 10FFFFH>, napisać fragment programu w asemblerze, który przekształci znak w formacie UTF-8 (4 bajty) na znak w formacie UTF-16 (2 słowa 16-bitowe). Położenie kodu źródłowego wskazuje zawartość rejestru ESI, a kod wynikowy powinien zostać wpisany do pamięci w miejsce określone przez zawartość rejestru EDI. Przyjąć kodowanie *mniejsze niżej* (ang. *little endian*).