

Kolokwium nr 1

Kolokwium nr 1

Zadania 1 z enauczania

1. W czterech kolejnych bajtach pamięci znajdują się liczby:

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

$L_E : 2A0037F2H$
 $B_E : F237002AH$

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. Podać zawartość rejestru BH w postaci liczby dziesiętnej po wykonaniu poniższych rozkazów:

```
mov    bh, 11  
xor    bh, 15
```

$$\oplus \begin{array}{l} 00001011 \\ 00001111 \\ \hline 00000100 \end{array} \Rightarrow BH = 4$$

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

$\begin{matrix} & a \\ \text{xor} & \quad \quad b \\ & \text{edi}, \text{esi} \end{matrix}$

W napisanym fragmencie nie można używać rozkazu xor.

$$\text{XOR: } \begin{array}{c|cc|c} & a & & 1 \\ b \backslash & 0 & 0 & \\ \hline 0 & 0 & 1 & \\ 1 & 1 & 0 & \end{array} \quad f(x) = a\bar{b} + \bar{a}b = (a+b)(\bar{a} + \bar{b}) =$$

mov ebx, edi
mov edx, esi
not edx
and ebx, edx
not edi
and edi, esi
or edi, ebx

mov ebx, edi
and ebx, esi
not ebx
or edi, esi
and edi, ebx

$a + b = OR$
 $a \cdot b = AND$

```
mov ebx, edi
and ebx, esi
not ebx
or edi, esi
and edi, ebx
```

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

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

Wynik dzielenia znajdować się będzie w rejestrze AX. Maksymalna wartość wyniku to $2^{16} - 1$, a podany fragment programu wygeneruje wynik = 2^{16} .

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

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

Pierwszy rozkaz wnuści na stos wartość odczytana z 32 najmiedzzych bitów rejestru 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
```

1	2
2	0
3	0
4	0
5	0
6	0
7	3
8	0
9	3
10	0
11	3
12	0
13	3
14	0
15	3
16	0
17	3
18	0
19	3
20	0
21	3
22	0
23	3
24	0
25	2
26	0
27	X 4
28	1
29	0
30	0
31	Q 3
32	0
33	0
34	0
35	0

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 ↴

obroty dd ≠ dup(?) rezerwacja 28 bajtów
 w pamięci

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

```
xor    eax,  eax  
sub    eax,  0FFFFFFF
```

OF - wskazuje na pomyślnie gdy dodawano licby U2

CF - 11- licby ber znakem

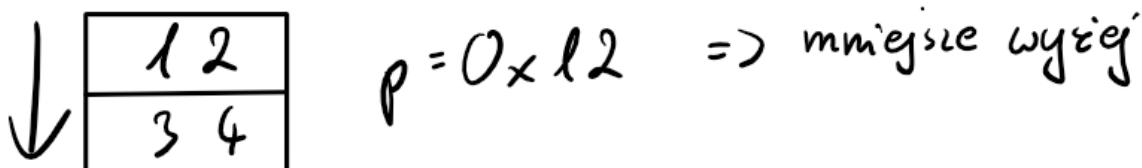
ZF - ustawiana na 1 gdy wynikiem jest 0

$$\text{sub ex, OFFFFFFFFFH} \Rightarrow 0 - (-l) = +l$$

$OF = 0 \quad CF = l \quad ZF = 0$

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*.

```
rcl ebx, 1  
jc ujemna  
rcr ebx, 1  
jmp koniec  
ujemna:  
neg ebx  
rcr ebx, 1  
koniec:
```

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

mov	dh,	15	00	0
xor	dh,	12	01	1
			10	1
			11	0

$00001111 \curvearrowright DH = 03H = 3$

00001100

00000011

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

lea esi, [esi + esi*8]

Zostanie pomnoione razy 9.

13. Na czym polega błąd w poniższym fragmencie programu:

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

Nie jest podane na ile bitach ma być zapisać litera 'A'. Jednym z poprawnych rozkazów jest „dword PTR”.

14. Rozkazy

```
push    ebx  
push    ecx
```

można zastąpić równoważną sekwencją:

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

Uzupełnić pola adresowe podanych wyżej rozkazów mov.

```
sub esp, 8  
mov [esp+4], ebx  
mov [esp], ecx
```

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

```
mov    eax, 0xFFFFFFFFH  
mov    ebx, 0xFFFFFFFFH  
imul  ebx
```

Wynik tego mnożenia wynosi w EDX:EAX.

EDX = 00000000H

EAX = 00000001H

16. Na czym polega błąd w poniższym fragmencie programu?

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

Liczba, której chcemy dać do obszaru pamięci o nazwie v2 jest za duża (o 4 bity).

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

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

ZF = 1, CF = 1, OF = 0
 $\hookrightarrow 0 + 1 = 1$ (najstarsze bity obu cyfr)

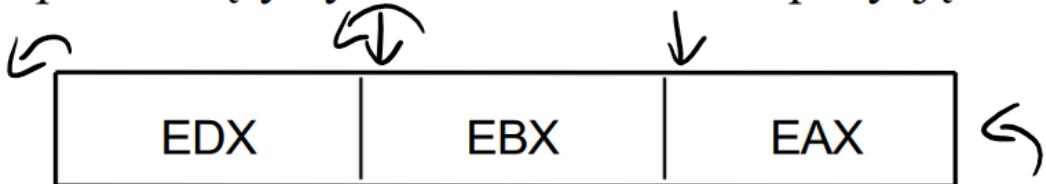
18. Podać liczbę, która zostanie wyświetlona na ekranie w wyniku wykonania poniższego fragmentu programu. Podprogram wyswietl32 wyświetla na ekranie w postaci dziesiętnej liczbę binarną zatrzytą w rejestrze EAX.

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

1	FE
2	00
3	FF
4	00
5	00
6	01
7	00

$EAX = 0FF00H = 16^3 \cdot 15 + 16^2 \cdot 15 = 65280$

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.



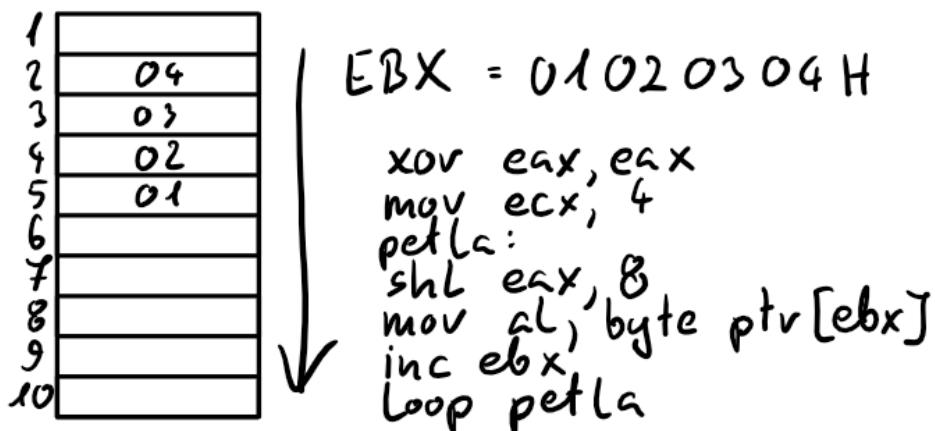
Wskazówka: wykorzystać rozkazy przesunięcia w lewo SHL (bity wychodzące z rejestru wpisywane są do CF) i RCL (zawartość CF wpisywana jest na najmłodszy bit rejestru, a bity wychodzące z rejestru wpisywane są do CF). Wykorzystać także rozkaz BT.

```
bt ebx, 31
rcl edx, 1
bt eax, 31
rcl ebx, 1
shl eax, 1
```

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

```
rol edx, 16
```

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.

```
xor dl, dl
mov cl, 32
petla:
rol eax, 1
jc dodaj
loop petla
jmp koniec
dodaj:
inc dl
loop petla
koniec:
mov cl, dl
```

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 $a \cdot 10 = a \cdot 8 + a \cdot 2$.

```
lea eax, [eax+4*eax]  
lea eax, [2*eax]
```

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

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

$$b = 1 \quad W = 2 \cdot 8 + 4 \cdot 2 + 2 + 10 \cdot l =$$

$$\omega = 2 \quad = 16 + 8 + 12 = 36 \text{ bajtów}$$

$$d = 4$$

$$q = 8$$

25. Na czym polega błąd w poniższym fragmencie programu?

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

Próbowujemy przypisać wartość 256 do pola gdzie maksymalnie można wpisać 255.

26. Wyjaśnić działanie poniższego fragmentu programu

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

Nieskończona pętla, która odejmuje 10 od wartości znajdującej się w rejestrze AX i zapisuje nową wartość do tego samego rejestru.

27. Podać zawartość rejestru EIP po wykonaniu poniższej sekwencji rozkazów

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

$$EDX = 347$$

$$\begin{aligned} [esp] &= EDX \\ EIP &= [ESP] \end{aligned} \quad \rightarrow EIP = 0000015BH$$

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

loop oblicz
oblicz: add dh, 7

dec ecx
jnx oblicz lub dec ecx
oblicz:
add dh, f

29. Na czym polega błąd w podanym niżej zapisie rozkazu

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

Występuje tu dwukrotne odwołanie się do pamięci.

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.

```
mov eax, OFFSET wskaznik  
mov wskaznik, eax
```

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] (=> mov edx,[ebx+esi])
```

$b = 1 \quad dd = 4 \quad 421 = 01A5$
 $w = 2 \quad dq = 8 \downarrow 0 \boxed{A5010000} \boxed{A6010000}$
 $\text{EDX} = 443 = 000001\text{B3H}$

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

```
mov eax, 125  
mov ebx, 10  
xor ecx, ecx  
petla:  
    xor edx, edx  
    div ebx  
    add cl, dl  
    cmp eax, 0  
    jnz petla
```

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

```

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

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

```

Wyświetlony tekst w okienku: „*o informatyka*”
 Nazwa okienka: „*infoinformatyka*”

0	ı
1	n
2	f
3	o
4	r
5	m
6	a
7	t
8	g
9	u
10	m
11	o
12	t
13	?
14	?
15	?
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	



34. W tablicy znaki znajdują 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	0xxxxxxx
80H	7FFH	110xxxxx 10xxxxxx
800H	FFFFH	1110xxxx 10xxxxxx 10xxxxxx

```

xor eax, eax           ; adres tekstu
xor ecx, ecx           ; ilosc bajtow
xor esi, esi           ; iterator
petla:
    mov dl, byte ptr [eax][esi]

```

```
cmp dl, 0
je koniec
rol dl, 1
jc jedynka
inc ecx
inc esi
jmp petla
jedynka:
rol dl, 2
jc dwie
inc ecx
add esi, 2
jmp petla
dwie:
inc ecx
add esi, 3
jmp petla
koniec:
add ecx, ecx
```

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 -> sład = adres kolejnego rozkazu, a ten = 5
dd 5
dd 7
jmp ciąg_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.

Treba przed „ret” ustawić add [esp], dword ptr 8 żeby przekonczyć te liczby.

35. 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.

```
mov edx, [ebx] [ecx*4]
mov ecx, 32
mov esi, 1
petla:
ror edx, 1
jc swieto
inc esi
loop petla
jmp koniec
swieto:
mov eax, esi
call wyswietl_EAX
inc esi
loop petla
koniec:
```

Ile bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze?

Odpowiedz proszę wyrazić jako liczbę w systemie dziesiętnym (np. 23).

s dw 3,-3,5,8 2·4
dd 'A' 4
dq 2 dup (0),0 3·8
t db 'abc' 3

39

Podany poniżej podprogram wykorzystuje jako argumenty liczby umieszczone bezpośrednio za rozkazem call, który wywołuje ten podprogram.

Podaj zawartość rejestru DX wyrażoną jako liczba dziesiętna (np. 15)

```
dodaj PROC
    mov ebx, [esp]
    add [esp], dword ptr 12
    mov dx, 0
    mov dh, [ebx+9]
    sub dh, [ebx+7]
    ret
dodaj ENDP
;

_main PROC
call dodaj
    dd 01020304h
    dd 01010101h
    dd 02020202h
jmp skok
```

DX
DH DL
dd = 4 bajty
 $DH = 02 - 01 = 01$
 $DX = 0100h$

ebx	0	D4
1	1	D3
2	2	G2
3	01	
4	01	G1
5	01	
6	01	G1
7	01	
8	02	G2
9	02	
10	02	G2
11	02	
12		
13		
14		



Ille bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze?

Odpowiedz proszę wyrazić jako liczbę w systemie dziesiętnym (np. 23).

```
v1 dw 'A',0 2.2
      dd 'ABCD' 4
      dq -1, 4 dup (0) 8.5
v3 db 10 dup (?) 10
```

b 1 } 58B
w 2 }
d 4 }
q 8 }

Podaj liczbę w systemie dziesiętnym (np. 12), która powinna zostać użyta jako indeks w adresowaniu indeksowym (linijka nr. 6) tak aby ustawić wartość -1 jako ostatni element tablicy destination.

Kat III. O/ tak aby ustawić wartość -1 jako ostatni element tablicy destination.

```
1.    .data
2.    destination      dw 44 dup (0DEADH)
3.    .code

4.    _main PROC
5.        mov         eax, dword PTR OFFSET destination
6.        mov         [eax + ____], word PTR -1
```

0	AD
1	DE
2	AD
3	DE
4	AD
5	DE
6	AD

84	AD
85	DE
86	AD
87	DE
88	
89	

word < +86

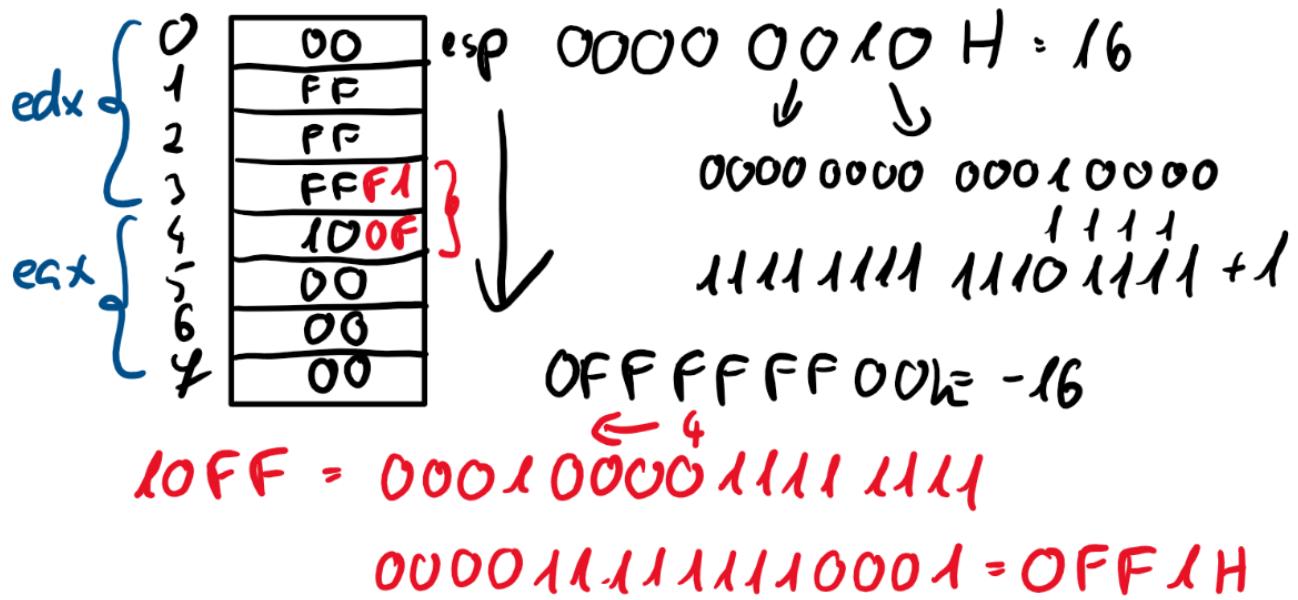
Po wykonaniu programu podaj zawartość zmiennej save.

```
1.    mov         esi, 0      DFFF > 1
2.    mov         ax, 1
3.    mov         dx, -1
4.    cmp         ax, dx
5.    ja          ptc
6.    mov         esi, 15      score = 15
7.    ptc:
8.        mov         eax, dword PTR OFFSET save
9.        mov         [eax], esi
```

Wynik powinien zostać podany w postaci liczby dziesiętnej (np. 12).

Jaką zawartość (w zapisie heksadecymalnym, zgodnym z asemblerem, np. 0FEEDh) ma rejestr **AX** po wykonaniu instrukcji?

1 push 16
2 push -16
3 rol word ptr [esp+3], 4
4 pop edx
5 pop eax



W programie asemblerowym został odczytany plik tekstowy zakodowany w UTF-16. Odczytany ciąg bajtów został umieszczony w buforze w pamięci operacyjnej. Wiedząc, że plik rozpoczyna się od ciągu 'łal' i w buforze w pamięci operacyjnej znajduje się m.in. ciąg bajtów jak na rysunku poniżej, wskaż brakujące bajty znajdujące się na początku pliku.

Adres	Wartość
6000001C	.
6000001B	.
6000001A	01
60000019	42
60000018	00
60000017	61
60000016	01
60000015	42
60000014	.
60000013	.

↑
y => LÉ

BOM
UTF-8 : 0xEF BB BF H
UTF-16 LE : 0xFF FE H
UTF-16 BE : 0xFE FF H

Uwaga: Wartości punktów kodowych dla polskich znaków:

'ł' - U+0142

Wybierz jedną odpowiedź:

FF FE

FE FF

EF BB BF

FF FE 00 00

00 00 FE FF

Podaj liczbę w systemie dziesiętnym (np.12), która powinna zostać użyta jako indeks w adresowaniu indeksowym (linijka nr. 6) tak aby ustawić wartość -1 jako ostatni element tablicy destination.

- | | | |
|----|-------------|-----------------------------------|
| 1. | .data | |
| 2. | destination | dd 20 dup (0DEADBEEFH) |
| 3. | .code | |
| 4. | _main PROC | |
| 5. | mov | eax, dword PTR OFFSET destination |
| 6. | mov | [eax + ____], dword PTR -1 |



Na procesorze zgodnym z architekturą x86, spod adresu 0x0023861A został skopiowany fragment pamięci o rozmiarze 2 bajtów z użyciem rozkazu `mov ax, word PTR [0023861AH]`. Wiadomo, że skopiowane dane są w formacie UTF-8 i reprezentują znak o konkretnym punkcie kodowym.

0x0023861C	1010 0101
0x0023861B	1101 0001
0x0023861A	1011 0000
0x00238619	1101 0100
0x00238618	1000 1110

Podaj wartość skopowanego punktu kodowego w formacie:

U+"Punkt Kodowy w postaci hexadecymalnej"

np. punkt kodowy dla symbolu "β" (03B2)₁₆ powinien zostać wprowadzony jako U+03B2.

Zakresy wartości punktów kodowych		Liczba kodo-wanych bitów	Reprezentacja w postaci UTF-8		
od	do				
0000	007F	7	0xxxxxx		
0080	07FF	11	110xxxxx	10xxxxxx	←
0800	FFFF	16	1110xxxx	10xxxxxx	10xxxxxx
10000	1FFFFFF	21	11110xxx	10xxxxxx	10xxxxxx

Na procesorze zgodnym z architekturą x86, spod adresu 0x0023861A został skopiowany fragment pamięci o rozmiarze 2 bajtów z użyciem rozkazu *mov ax, word PTR [0023861AH]*. Wiadomo, że skopiowane dane są w formacie UTF-8 i reprezentują znak o konkretnym punkcie kodowym.

0x0023861C	1010 0101
0x0023861B	1101 0001
0x0023861A	1011 0000
0x00238619	1101 0100
0x00238618	1000 1110

→ $AX = OD1B0H$

~~1101 0001
1011 0000~~
440H

Podaj wartość skopiowanego punktu kodowego w formacie:

U+"Punkt Kodowy w postaci hexadecymalnej"

U+0440

np. punkt kodowy dla symbolu "β" (03B2)₁₆ powinien zostać wprowadzony jako U+03B2.

Zakresy wartości punktów kodowych		Liczba kodowanych bitów	Reprezentacja w postaci UTF-8		
od	do		0xxxxxx	110xxxxx 10xxxxxx	1110xxxx 10xxxxxx 10xxxxxx
0000	007F	7	0xxxxxx		
0080	07FF	11		110xxxxx 10xxxxxx	
0800	FFFF	16			1110xxxx 10xxxxxx 10xxxxxx
10000	1FFFFFF	21			11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Na etapie debugowania programu programista zatrzymał swój program na linii 6 przy pierwszym wykonaniu pętli (skok jnz nie został jeszcze wykonany). Podaj zawartość rejestru EAX w postaci hexadecymalnej, która się wyświetli w debuggerze przy podglądzie rejestrów.

1.	mov	ebx, 32
2.	mov	edx, 2
3.	ptl:	
4.	mov	eax, 514
5.	div	ebx
6.	sub	edx, 1
7.	jnz	ptl

$$\begin{aligned}EBX &= 32 \\EDX &= 2 \\EAX &= 514\end{aligned}$$

Odpowiedź podaj w formacie 32-bitowym, tak jak w asemblerze: np. liczba $(10)_{10}$ powinna zostać wprowadzona jako: 0000000Ah.

$$EAX = \frac{EDX \cdot EAX}{32} \Rightarrow \text{przesuwając o } 5 \text{ o pozycję}$$

0010:0000|0000|0000|0000|0000|0010p000 0010
0001:0001|0000|0000|0000|0000|0000 00010000
0000: 10000010H = EAX

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.

$$\begin{aligned}P &\quad q \\ \text{xor } eax, edx \quad EAX &= p \times 0Rq = q \times 0Rp \\ \text{xor } edx, eax \quad EDX &= q \times 0Rq \times 0Rp = p \\ \text{xor } eax, edx \quad EAx &= p \times 0Rp \times 0Rq = q\end{aligned}$$

Wskazówki: określić zawartość k-tego bitu rejestrów 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 $x \oplus x = 0$.

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

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

Wynik divedenia musi się zmieścić w registerze AX, a przy divedeniu przez 1 dostawemy wynik 2³².

2. (1 pkt.) Określić postać komunikatu wyświetlanego przez funkcję MessageBox po wykonaniu poniższego fragmentu programu.

```
b 1  
w 2  
d 4  
9 8  
tekst dw 'ar', 'ch', 'it', 'ek', 'tu', 'ra', 0  
push 0  
push OFFSET tekst ; tytuł  
lea eax, dword ptr [tekst+6]  
push eax ; treść  
push 0  
call _MessageBoxA@16
```

LÉ ↓

0	r
1	a
2	h
3	c
4	t
5	i
6	u
7	e

9	u
10	t
11	a
12	r
13	o
14	o
15	

tytuł = rachunki kwater
tekst = kwater

3. (4 pkt.) W tablicy input zadeklarowano łańcuch 16-bitowych znaków UTF-16, zakończony słowem o wartości 0DEADh (kodowanie znaków LittleEndian).

Podaj fragment programu w 32-bitowym asemblerze, który zbuduje datagram w tablicy output. Datagram wynikowy konstruowany jest na podstawie tablicy input, w taki sposób, że w tablicy output mają się znaleźć tylko i wyłącznie znaki z podstawowego zbioru ASCII. Dodatkowo, na końcu utworzonego datagramu należy umieścić 8-bitową sumę kontrolną CRC, zdefiniowaną jako suma modulo wszystkich kolejnych, bajtowych danych znajdujących się w wynikowym datagramie.

```
xor eax, eax  
xor esi, esi  
xor edi, edi  
mov ebp, offset input  
mov ebx, offset output
```

```

petla:
    mov dx, word ptr [ebp+2*esi]
    inc esi
    cmp dx, 0DEADh
    je koniec
    cmp dx, 7Fh
    ja nastepne
    mov [ebx+2*edi], dx
    inc edi
    xor al, dl
nastepne:
    jmp petla
koniec:
    mov [ebx+2*edi], al

```

1. (1 pkt.) W pewnym programie została zdefiniowana zmienna

wskaznik db 4 dup (?)

Napisać fragment programu w 32-bitowym asemblerze, który wpisze do tej zmiennej adres komórki pamięci, w której znajduje się ta zmienna

*mov eax, OFFSET wskaźnik
mov dword ptr wskaźnik, eax*

1. (1 pkt.) Rozkaz (błędny) MOV ebp, dword PTR bx miał w zamierzeniu autora programu przesyłać liczbę z rejestru BX do rejestru EBP. Podać poprawną postać tej operacji przy założeniu, że w rejestrze BH znajduje się liczba ze znakiem (U2).

movsx ebp, bx

2. (1 pkt.) Określić postać komunikatu wyświetlanego przez funkcję MessageBox po wykonaniu poniższego fragmentu programu.

```

tekst dw 'a', 'r', 'c', 'h', 'i', 't', 'e', 'k', 't', 'u', 'r', 'a', 0
- - - - - - - - - -
    push 0
    push OFFSET tekstu
    lea   eax, [tekstu+3]
    push eax
    push 0
    call _MessageBoxA@16

```

w tytule wyświetli się „a”, a pole będące puste.

Podaj zawartość rejestru **BX** wyrażoną jako liczba dziesiętna.

```
dodaj PROC
    mov edi, [esp]
    add [esp], dword ptr 8
    mov ebx, 0
    mov bh, [edi+5]
    add bh, [edi+3]
    ret
dodaj ENDP
;
call dodaj
dd 01020304h
dd 01010001h
jmp skok
```

0	04
1	03
2	02
3	01
4	01
5	00
6	01
7	01
8	
9	
10	

edi

$$00 + 01 = 01$$

$$BX = 0100H = 256$$

Podaj zawartość rejestru **ecx** (jako wartość dziesiętną, bez zer wiodących, np. 123) po wykonaniu następującego fragmentu kodu.

```
mov ecx, -1           ; ecx = 0FFFFFOO18h
mov cx, 22
skok:
    sub cx, 11   22 -> 11 -> 0
    je et2
    call skok [esp] = et2
et2:
    mov ecx, [esp]      ; ecx = -et2
    neg ecx
    lea ecx, [ecx+et2+1] ; ecx = -et2 + et2 + 1 = 1
```

Na procesorze zgodnym z architekturą x86, spod adresu 0xAB11FAAC został skopiowany fragment pamięci o rozmiarze 2 bajtów z użyciem rozkazu *mov ax, word PTR [AB11FAACH]*. Wiadomo, że skopiowane dane są w formacie UTF-8 i reprezentują znak o konkretnym punkcie kodowym.

A memory dump table and a hand-drawn binary representation of the character 'C' are shown. The table lists memory addresses from 0xAB11FAAA to 0xAB11FAAE, with their corresponding hex values. A large downward arrow points to the row for 0xAB11FAAC. To the right, the binary representation '11001111 10000000 C 0' is written in red, with arrows indicating the mapping from the table row to the binary digits. Below the binary digits, the character 'U+03C0' is written.

0xAB11FAAA	0010 0101
0xAB11FAAB	1110 0101
0xAB11FAAC	1000 0000
0xAB11FAAD	1100 1111
0xAB11FAAE	1010 0110

11001111
10000000
C 0
U+03C0

Podaj wartość skopiowanego punktu kodowego w formacie:

U+"Punkt Kodowy w postaci hexadecymalnej"

np. punkt kodowy dla symbolu "β" (03B2)₁₆ powinien zostać wprowadzony jako U+03B2.

Zakresy wartości punktów kodowych		Liczba kodowanych bitów	Reprezentacja w postaci UTF-8
od	do		
0000	007F	7	0xxxxxx
0080	07FF	11	110xxxxx 10xxxxxx
0800	FFFF	16	1110xxxx 10xxxxxx 10xxxxxx
10000	1FFFFFF	21	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Na procesorze zgodnym z architekturą x86, spod adresu 0xFFFF1425 został skopiowany fragment pamięci o rozmiarze 2 bajtów z użyciem rozkazu `mov ax, word PTR [FFFF1425H]`. Wiadomo, że skopiowane dane są w formacie UTF-8 i reprezentują znak o konkretnym punkcie kodowym.

0xFFFF1423	0010 0101
0xFFFF1424	1100 0101
0xFFFF1425	1000 0101
0xFFFF1426	1100 0100
0xFFFF1427	0010 0110

Podaj wartość skopiowanego punktu kodowego w formacie:

U+"Punkt Kodowy w postaci hexadecymalnej"

np. punkt kodowy dla symbolu "β" (03B2)₁₆ powinien zostać wprowadzony jako U+03B2.

Zakresy wartości punktów kodowych		Liczba kodo-wanych bitów	Reprezentacja w postaci UTF-8		
od	do				
0000	007F	7	0xxxxxx		
0080	07FF	11	110xxxxx	10xxxxxx	
0800	FFFF	16	1110xxxx	10xxxxxx	10xxxxxx
10000	1FFFFFF	21	11110xxx	10xxxxxx	10xxxxxx

Na etapie debugowania programu programista zatrzymał swój program na linii 6 przy pierwszym wykonaniu pętli (skok `jnz` nie został jeszcze wykonany). Podaj zawartość rejestru **EAX** w postaci hexadecymalnej, która się wyświetli w debuggerze przy podględzie rejestrów.

```

1.    mov        ebx, 32
2.    mov        edx, 3
3.    ptl:
4.          mov        eax, 129
5.          div        ebx 32
6.          sub        edx, 1
7.          jnz        ptl

```

Odpowiedź podaj w formacie 32-bitowym, jak w asemblerze: np. liczba (10)₁₀ powinna zostać wprowadzona jako: 0000000Ah.

0011 : 0000|0000|0000|0000|0000|0000|0000|0000|1000|0001
 0001 : 0001|1000|0000|0000|0000|0000|0000|0000|0000|0100
EAX = 18000004H

Podaj zawartość rejestru **ebx** (jako wartość dziesiętną, bez zer wiodących, np. 123) po wykonaniu następującego fragmentu kodu:

```
    mov  ebx,0ffffFFFFFFh } EBX = FFFF 0002 H
    mov  bx,2
etykieta:
    sub  bx,1 -> 2-1: l-l=0
    jz   dalej
    call etykieta -> [esp]=dalej
dalej:
    lea   ebx,[dalej] ebx = dalej
    sub  ebx,[esp] -> ebx = dalej - dalej = 0
```

1. (1 pkt.) W pewnym programie została zdefiniowana tablica

obszar dw 2 dup (?)

Napisać fragment programu w asemblerze, który wpisze do tej tablicy adres aktualnie wykonywanej instrukcji tzn. pierwszego (i być może jedynego) Pan/Pani rozkazu.

mov eax, \$

mov dword ptr obszar, eax

2. (1 pkt.) Jaka wartość zostanie wpisana do rejestru EAX po wykonaniu podanego niżej rozkazu (wynik podać w postaci 8-cyfrowej liczby szesnastkowej)?

```
pqr      dw    257, 129, 65
```

```
- - - - - - - - - - - - - - - -
```

```
mov      ebx, OFFSET pqr
```

```
lea      edi, [ebx+1]
```

```
mov      eax, [edi]   EAX = 41008101
```

$$(257)_{10} = 101H$$

b 1

$$(129)_{10} = 81H$$

w 2

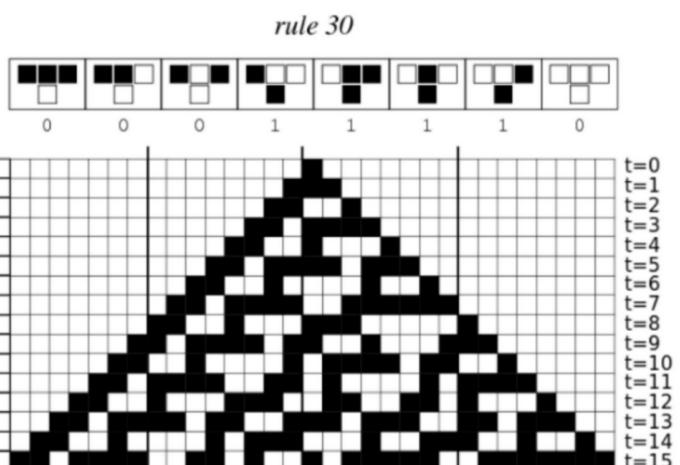
$$(65)_{10} = 41H$$

0	01
1	01
2	81
3	00
4	41
5	00
6	6
7	4
8	8
9	9
10	10

4. (4 pkt.) Podaj fragment kodu w 32-bitowym asemblerze x86 realizujący prosty automat komórkowy Wolframa działający wg reguły 30 (obrazek po prawej). Automat modyfikuje poszczególne bity kolejnych 4 bajtowych rzędów zgodnie ze stanem danego bitu i bitów sąsiednich w rzędzie poprzednim, tak jak przedstawiono na załączonym rysunku. Załóż, że w pamięci statycznej zdefiniowano następujące obszary:

```
starting_row db 0h, 0h, 80h, 0h
next_rows db 15*4 dup(?)
```

Kod powinien sekwencyjnie uzupełnić obszar next_rows właściwymi wartościami.



```

mov eax, dword ptr [starting_row]
bswap eax           ; w pamięci jest na odwrot niż jak chcemy
xor edi, edi        ; licznik wierszy
next_row:
xor edx, edx
mov ecx, 32
set_row:
mov bl, dl          ; tymczasowa kopia
and bl, 0111b        ; maska
jz bit_set
cmp bl, 4
ja bit_set

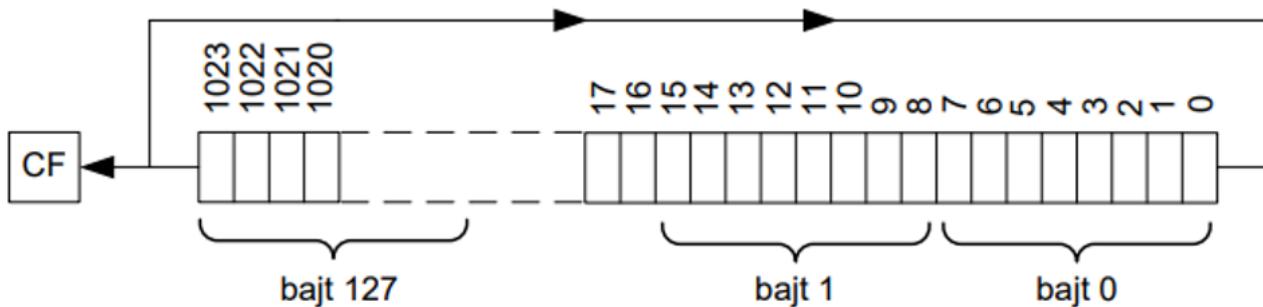
```

```

bts edx, 1
bit_set:
    ror eax, 1          ; zrodlowy
    ror edx, 1          ; docelowy
loop set_row
linia_gotowa:
    mov eax, edx
    bswap edx
    mov dword ptr [next_row+4*edi], edx
    inc edi
    cmp edi, 15
    jne next_row

```

37. 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:

- Przyjąć, że tablica *rejestr1024* została wcześniej zdefiniowana w sekcji danych programu jako tablica 128-bajtowa.
- Wykorzystać m.in. rozkazy SHL i RCL.
- 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.

```
podprogram PROC
    push ebp
    mov ebp, esp
    push esi
    push ebx          ; adres tablicy
    mov ecx, 128      ; ilosc petli
    mov ebx, offset tablica1024
    xor esi, esi      ; iterator tablicy
    xor eax, eax      ; pomocnicze
    clc              ; wyczyszczenie CF

petla:
    mov al, byte ptr [ebx+esi]
    jc zmien_jeden
    btr ax, 8
    jmp dalej
zmien_jeden:
    bts ax, 8
    jmp dalej
dalej:
    rol al, 1
    mov byte ptr [ebx+esi], al
    inc esi
    loop petla
koniec:
    mov al, byte ptr [ebx]
    jc ustaw_jeden
    btr ax, 8
    jmp epilog
ustaw_jeden:
    bts ax, 8
epilog:
    mov byte ptr [ebx], al
    pop ebx
    pop esi
    pop ebp
podprogram ENDP
```

Napisać podprogram, który podzieli liczbę w kodzie U2 zawartą w rejestrze EAX przez -2. Resztę z dzielenia (0 lub +1) należy wpisać do znacznika CF, a iloraz również do rejestru EAX.

Dzielenie powinno być przeprowadzone w taki sposób by dzielna, dzielnik, iloraz i reszta spełniały poniższy związek:

$$\text{dzielna} = \text{iloraz} * \text{dzielnik} + \text{reszta}$$

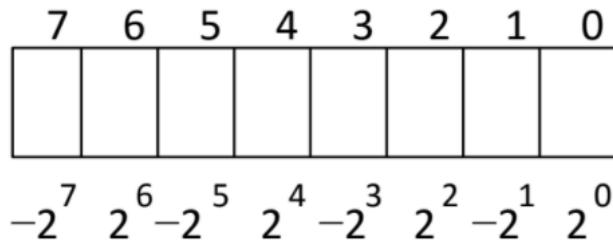
Reszta z dzielenia musi być liczbą nieujemną (0 lub 1).

Dzielenie przeprowadzić za pomocą rozkazu SAR (nie stosować DIV ani IDIV). Rozkaz SAR nie zmienia bitu znaku (bit znaku jest powielany).

```
dziel PROC
    sar eax, 1
    pushf           ; przechowanie znacznikow
    neg eax         ; neg wpisuje 0 do CF jesli eax = 0 inaczej wpisuje
    1
    popf            ; odtworzenie znacznikow
    ret
dziel ENDP
```

W systemie binarnym o podstawie ujemnej -2 (kod minus dwójkowy, zob. rys. liczby 8-bitowej) wartość liczby określa wyrażenie

$$w = \sum_{i=0}^{m-1} b_i \cdot (-2)^i = \sum_{i \text{ parzyste}} b_i \cdot 2^i - \sum_{i \text{ nieparzyste}} b_i \cdot 2^i$$



Zamiana liczby w kodzie U2 na liczbę w kodzie minus dwójkowym polega na wielokrotnym dzieleniu przez -2 i składaniu uzyskiwanych reszt (0 lub 1). Dzielenie kończy się po wystąpieniu ilorazu równego 0.

$$\text{dzielna} = \text{iloraz} * \text{dzielnik} + \text{reszta}$$

Przykład: $(9)_{10} = (1001)_{U2} = (11001)_{\text{minus dwójkowy}}$

$$+9/(-2) = -4, \text{ reszta} = 1$$

$$-4/(-2) = 2, \text{ reszta} = 0$$

$$+2/(-2) = -1, \text{ reszta} = 0$$

$$-1/(-2) = 1, \text{ reszta} = 1$$

$$+1/(-2) = 0, \text{ reszta} = 1$$

Napisać podprogram, który zamieni liczbę w kodzie U2 w rejestrze EAX na liczbę w kodzie minus dwójkowym i wynik wpisze do rejestrów EAX. Dzielenie przez -2 przeprowadzić za pomocą opisanego wcześniej podprogramu.

```
minus_dwojekw PROC
    push ebp
    mov ebp, esp
    mov eax, [ebp+8]           ; liczba w kodzie u2
    xor edx, edx
    xor ecx, ecx

    zamiana:
        or eax, eax
        jz gotowe
        call dziel
        adc edx, 0

gotowe:
    pop ebp
    ret
```

```

inc ecx
ror edx, 1
jmp zamiana

gotowe:
rol edx, cl
mov eax, edx

pop ebp
ret

minus_dwojkowy ENDP

```

38. Napisz podprogram `zapisz5bitow`, który pobiera 5 najmłodszych bitów z rejestru AL i zapisuje je w pamięci począwszy od bajtu o adresie podanym w rejestrze EDI i bitu o numerze (7 – 0) podanym w rejestrze CL. Pozostałe bity w bieżącym bajcie i sąsiednim powinny pozostać niezmienione.

```

zapisz5bitow PROC
push ebx

mov dx, word ptr [edi]      ; do DX dwa bajty z pamiec
and ax, 1Fh                ; wyzerowanie wszystkich oprocz najmłodszych
5
mov bx, OFFE0h              ; maska
rol ax, cl                  ; AX w lewo o tyle ile CL
rol bx, cl
and dx, bx                  ; zerujemy odpowiednie bity
or dx, ax                   ; na odpowiednich bitach ustawiamy wartosci
z AX
mov word ptr [edi], dx      ; zwracamy 2 bajty z powrotem do pamieci

pop ebx
ret

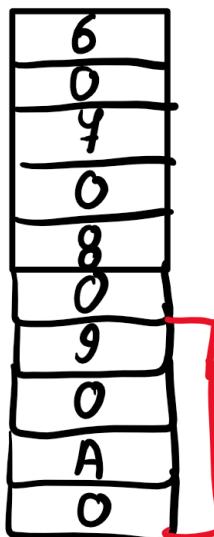
zapisz5bitow ENDP

```

pon dd 0,1	b 1
wt dd 2,3,4,5	w 2
sr dw 6,7,8,9,10,11	d 4
	g 8

mov edi, wt-pon
mov eax, dword ptr sr[edi-2]

$$wt - pon = 8$$



$$EAX = 000A0009H$$

Twój indeks jako liczba szesnastkowa. Określić zawartość znaczników OF, CF i ZF po wykonaniu rozkazów

shl eax, 0Bh

$$B = 11$$

sub eax, 4000 0000h

$$EAX = 0018\ 440FH$$

$$\begin{array}{r}
 0000\ 0000\ 0001\ 1000\ 0100\ 0100\ 0000\ 0111 \\
 1100\ 0010\ 0010\ 0000\ 0011\ 1000\ 0000\ 0000 \\
 - 0100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
 \hline
 1000\ 0010\ 0010\ 0000\ 0011\ 1000\ 0000\ 0000
 \end{array}$$

$$ZF = 0 \quad CF = 0 \quad OF = 0$$

Znacznik ZF ustawia się na 1 gdy wynikiem operacji arytmetycznej jest 0.

Znacznik CF ustawia się na 1 gdy w operacji arytmetycznej wystąpiła prośba o pożyczkę. Wskazuje on na przeniesienie przy dodawaniu liczb bez znaku. Procesor ustawia te znacznik na podstawie wartości przeniesienia, które powstaje przy dodawaniu najstarszych bitów obu liczb.

Znacznik OF ustawia się na 1 wówczas, gdy dodanie dwóch liczb z różnymi znakami (w kodzie U2) daje wynik nie mieszczący się w pierwszym argumencie operacji.

Przykładem jest dodawanie 127 do 127 używając 8-bitowych rejestrów. Wynikiem jest $1111\ 1110b$, które w U2 przedstawia liczbę -2, liczbę negatywną. Negatywna suma dwóch pozytywnych operatorów jest overflowem, tak samo na odwrót.

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

7	6	5	4	3	2	1	0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}

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:

- Liczbę w rejestrze AL mnożymy przez 10 używając rozkazu mnożenia dwóch liczb 8-bitowych – wynik mnożenia zostaje wpisany do rejestrów AX.
- Liczba wpisana do rejestrów AH stanowi wartość kolejnej cyfry liczby dziesiętnej, zaczynając od najstarszej.
- Powtarzamy opisane operacje wymaganą ilość razy.
- Zamieniamy uzyskane wartości na kody ASCII i wyświetlamy na ekranie za pomocą funkcji *write*.

```
.data
wynik db '0', '.', 3 dup (0)

.code
_main PROC
    mov esi, 2
    mov al, 11100000b
    mov bl, 10          ; mnozenie *10

    petla:
        mul bl
        add ah, 30H        ; zamiana na ASCII
        mov wynik[esi], ah
        mov ah, 0
        inc esi
        cmp esi, 5
        je koniec
        jmp petla

koniec:
    push 5
    push offset wynik
    push 1
    call __write

    push 0
    call _ExitProcess@4

_main ENDP
END
```

40. Napisać podprogram w asemblerze, który wyświetli na ekranie zawartość rejestru AL w postaci liczby dziesiętnej, wykorzystując nizej 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 rejestrów EAX (z wyjątkiem AL).
- b. Zbadać czy liczba umieszczona na bitach 3 – 0 rejestrze AH jest większa od 4, jeśli tak, to do rejestrów EAX dodać liczbę 300H.
- c. Zbadać czy liczba umieszczona na bitach 7 – 4 rejestrze AH jest większa od 4, jeśli tak, to do rejestrów 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 znajdującej 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*.

podprogram PROC

```
push ebp
mov ebp, esp

mov ecx, 8 ; licznik iteracji petli
xor edx, edx

petla:
    mov dl, ah
    and dl, 0Fh
    cmp dl, 4
    jae dodaj1
    jmp dalej

dodaj1:
    add eax, 300h

dalej:
    mov dl, ah
    shr dl, 4
    and dl, 0Fh
    cmp dl, 4
    ja dodaj2
    jmp zrobLoop

dodaj2:
    add eax, 3000h

zrobLoop:
    shl eax, 1
    loop petla

    mov edx, eax
```

```

shr edx, 8
ror edx, 8

add dl, 30H
mov [liczba], dl
and dl, 00h
rol edx, 4

add dl, 30H
mov [liczba+1], dl
rol edx, 4
and dl, 00h

add dl, 30H
mov [liczba+2], dl

push 3
push offset liczba
push 1
call __write
add esp, 12
pop ebp
ret

```

podprogram ENDP

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.

$$\begin{array}{lll} \{ESI\} > \{EDI\} & \Rightarrow & CF = 0 \text{ i } ZF = 0 \\ \{ESI\} = \{EDI\} & \Rightarrow & CF = 0 \text{ i } ZF = 1 \\ \{ESI\} < \{EDI\} & \Rightarrow & CF = 1 \text{ i } ZF = 0 \end{array}$$

Uwagi:

- Zapisy $\{ESI\}$ i $\{EDI\}$ oznaczają, odpowiednio, wartości liczb wskazywanych przez rejesty ESI i EDI.
- Liczby mogą mieć niejednakową liczbę cyfr.
- Operację porównania przeprowadzić bez konwersji obu liczb na postać binarną.

```
.data
liczba1 db '1', '2', '3', '4', '5', 0
rozmiar1 db $ - liczba1
liczba2 db '2', '3', '4', '5', 0
rozmiar2 db $ - liczba2
```

```
.code
_main PROC
    mov esi, offset liczba1
    mov edi, offset liczba2
    xor eax, eax
    mov al, [rozmiar1]
    mov ah, [rozmiar2]
    cmp al, ah
    ja wieksza_pierwsza
    jb wieksza_druga

    xor ebp, ebp          ; iterator liczb
    mov cl, [rozmiar1]

czy_rowne:
    mov dl, [esi+ebp]
    cmp dl, [edi+ebp]
    inc ebp
    ja wieksza_pierwsza
    jb wieksza_druga
    loop czy_rowne
    clc
    xor edx, edx
    jmp koniec

wieksza_pierwsza:
    clc
    mov edx, 2
    sub edx, 1
    jmp koniec

wieksza_druga:
    stc
    mov edx, 2
    sub edx, 1
    jmp koniec

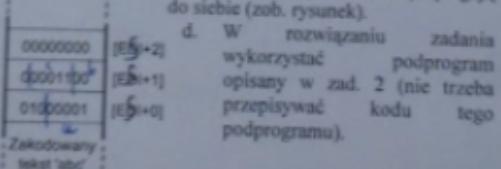
koniec:
```

1. (6 pkt.) W pewnym programie przetwarzane są teksty, w których występują wyłącznie małe litery alfabetu łacińskiego i znaki spacji (odstępu). W celu zmniejszenia obszaru pamięci zajmowanego przez te teksty przyjęto kodowanie wg poniższej tabeli.

Znak	Kod ASCII	Kod skrócony
koniec ciągu	0000 0000 (=0)	00000 (0)
a	01100001 (=61H)	00001 (1)
b	01100010 (=62H)	00010 (2)
c	01100011 (=63H)	00011 (3)
---	---	---
z	0111 1010 (=7AH)	11010 (26)
spacja	0010 0000 (=20H)	11011 (27)

Napisz podprogram krótszy_na_ASCII w asemblerze, który zamieli tekst w wersji skompresowanej (wg powyższej tabeli) na tekst w standardzie ASCII. Przyjąć następujące założenia:

- a. Adres początkowy tekstu skompresowanego podany jest w rejestrze ESI, a adres obszaru docelowego podany jest w rejestrze EDI.
 - b. Tekst źródłowy zakończony jest liczbą 0 w postaci 5-bitowej, a tekst wynikowy kończy się bajtem o wartości 0.
 - c. W obszarze źródłowym kolejne ciągi 5-bitowe przylegają



- W rozwiązyaniu zadania wykorzystać podprogram opisany w zad. 2 (nie trzeba przepisywać kodu tego podprogramu).

```
.model flat
public _main
extern _ExitProcess@4 :PROC

.data

skompresowane db 01000001b,00001100b,01010010b,11000000b,00000001b
wynik db 420 dup (0FFH)

.code

_main PROC

    mov esi, offset skompresowane
    mov edi, offset wynik
    mov eax, 0
    ;ebx:al = 40 bitow = 5 bajtow = 8 liter

    mov al, [esi]
    mov ebx, dword ptr [esi+1]

    mov ecx, 8
    petla:
        push ax
```

```
and al, 00011111b
cmp al, 27
jz spacja
cmp al, 0
jz koniec
add al, 60h
jmp dalej
spacja:
mov al, 20h
dalej:
mov [edi], al
inc edi
pop ax

push ecx
mov ecx, 5
o5bitow:           ;petla ktora przesuwa wyrazenie o 5 bitow w prawo
    bt ebx, 0
    rcr al, 1
    shr ebx, 1
    loop o5bitow
pop ecx

loop petla

koniec:
pop ax
mov [edi], byte ptr 0
inc edi

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

39. Zakładając, że został zdefiniowany następujący obszar danych:

```

suma_kontrolna    db ? ; miejsce na
przechowanie sumy kontrolnej
znaki db 1, 1, 1, 5 ; kolumna 2 tablicy
znaki zgodna z tabela 1 na końcu zadania
dw 0               ; wartość z kolumny 3
tabeli 1
db 1, 1, 2, 4
dw 15
----- ; tutaj
dalsza część tablicy
db 5, 1, 1, 1
dw 10

```

- a. dopisz do poniższego kodu fragment kodu w asemblerze 32-bitowym, który dla znaku ASCII (umieszczonego w rejestrze AL) z dopuszczalnego alfabetu ('0-9' oraz 'A-Z' bez 'O', łącznie 35 znaków) wyznaczy kod paskowy BC412 i umieści go w pamięci pod adresem wskazanym przez EDI. Najstarsze, niewykorzystywane bity powinny zostać wyzerowane. Pasek należy interpretować jako bit ustawiony, zaś przerwę jako bit wyzerowany. Przykładowo dla znaku '0' wyznaczona wartość jest równa (binarnie) 0000 1010 1010 0000.

```

mov esi,OFFSET lancuch ; adres obszaru z
                        ; tekstem do zakodowania
mov edi, OFFSET bufor ; adres na kody w
                        ; BC412
mov ecx,8              ; liczba znaków w
                        ; buforze do zakodowania
mov suma,0 ; inicjalna suma kontrola
                        ; ustwiona na 0

nastepny_znak:
..... ; tutaj wstaw swój kod
loop nastepny_znak

```

- b. dopisz fragment kodu w asemblerze 32-bitowym, do wyznaczania wartości dla sumy kontrolnej łańcucha (nie trzeba kodować wartości sumy na kod kreskowy BC412!).

Uwagi:

Kod kreskowy zastępuje tzw. znaki alfabetu obrazem z na przemian ulożonych pasków i wolnych przestrzeni pomiędzy nimi (patrz rys. poniżej). Jednym z typów kodu kreskowego jest kod BC412. Alfabetem kodu BC412 (czyli dopuszczalnych znaków) są cyfry 0-9 oraz wielkie litery A-Z (oprócz 'O'), którą zastępuje się znakiem zer.



K O L O K W I U M 1 A K 0

Kodowanie odbywa się poprzez zmienną odległość pustego miejsca między kolejnymi paskami.

Znak zwykły kodowany jest jako kod o szerokości 12 pasków. Na szerokość składa się czterokrotna kombinacja pasek + przestrzeń, gdzie szerokość przestrzeni podaje kolumna 2 w tabeli 1. Szerokość paska jest zawsze równa 1.

Np. dla znaku zera '0' wartość ta wynosi 1, 1, 1, 5. Oznacza to, że znak '0' ma postać:

pasek, przestrzeń, pasek, przestrzeń, pasek,
przestrzeń, pasek, 5x przestrzeń

Całość w sumie zajmuje 12-krotność szerokości paska (4 paski + 8 krotność paska w postaci pustych przestrzeni pomiędzy).

Wartość znaku przypisanego do **cyfry kontrolnej** ustala się jako sumę wartości znaków podanych w tabeli 1 (kolumna 3) podzieloną modulo 35.

Tabela 1. Wartości kodowe znaków alfabetu

Znak	Kodowanie	Wartość
0	1, 1, 1, 5	0
1	1, 1, 2, 4	15
2	1, 1, 3, 3	17
3	1, 1, 4, 2	29
4	1, 1, 5, 1	11
5	1, 2, 1, 4	33
6	1, 2, 2, 3	19
7	1, 2, 3, 2	21
8	1, 2, 4, 1	8
9	1, 3, 1, 3	2
A	1, 3, 2, 2	7
B	1, 3, 3, 1	25
C	1, 4, 1, 2	20
D	1, 4, 2, 1	22
E	1, 5, 1, 1	9
F	2, 1, 1, 4	30
G	2, 1, 2, 3	3
H	2, 1, 3, 2	6
I	2, 1, 4, 1	27
J	2, 2, 1, 3	16
K	2, 2, 2, 2	24
L	2, 2, 3, 1	4

.686

.model flat

```

public _main
extern _ExitProcess@4 : PROC

```

```

.data
suma db ?
znaki    db 1, 1, 1, 5
          dw 0
          db 1, 1, 2, 4
          dw 15
          db 1, 1, 3, 3
          dw 17
          db 1, 1, 4, 2
          dw 29
          db 1, 1, 5, 1
          dw 11
          db 1, 2, 1, 4
          dw 33
          db 1, 2, 2, 3
          dw 19
          db 1, 2, 3, 2
          dw 21
lancuch db '0', '3', '6', '2'
bufor dw 20 dup (?)
mnoznik db 6

.code
_main PROC
    mov esi, offset lancuch
    mov edi, offset bufor
    mov ecx, 4
    mov suma, byte ptr 0

    xor edx, edx           ; temporary
    xor ebx, ebx           ; iterator lancuch input

nastepny_znak:
    xor eax, eax           ; temp
; przekodowanie tego co jest w buforze na numer wiersza w tablicy
    mov al, [esi+ebx]
    inc ebx
    cmp al, 'A'
    je litery
    sub al, '0'            ; w AL indeks w tablicy
    jmp dalej

litery:

```

```
sub al, 'A'

dalej:
    mul mnoznik           ; EAX = 6*EAX
    mov eax, dword ptr [znaki+eax]

; zamiana wiersza w tablicy na binarna reprezentacje
    mov dx, 1              ; temporary
    push ecx               ; wartosc dla glownej petli

    mov cl, al              ; pierwsze od lewej kodowanie
    add cl, 1               ; miejsce na 1
    shl edx, cl
    bts edx, 0              ; ustawienie jedynki

    ror eax, 8
    mov cl, al              ; pierwsze od lewej kodowanie
    add cl, 1               ; miejsce na 1
    shl edx, cl
    bts edx, 0              ; ustawienie jedynki

    ror eax, 8
    mov cl, al              ; pierwsze od lewej kodowanie
    add cl, 1               ; miejsce na 1
    shl edx, cl
    bts edx, 0              ; ustawienie jedynki

    ror eax, 8
    mov cl, al              ; pierwsze od lewej kodowanie
    shl edx, cl

    mov [edi], dx
    add edi, 2

    pop ecx
    loop nastepny_znak

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