

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

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
```

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  

```

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, 0FFFFFFFH

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 równoważną 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, 0xFFFFFFFFH  
mov    ebx, 0xFFFFFFFFH  
imul   ebx
```

16. Na czym polega błąd w poniższym fragmencie programu?

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

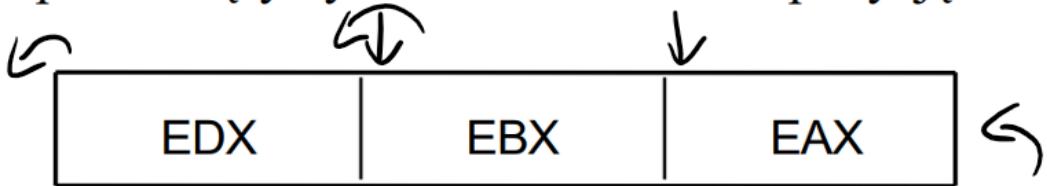
17. Określić zawartości znaczników OF, ZF i CF po wykonaniu podanego niżej fragmentu programu.

```
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 zasadztą 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 rejestr 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	0xxxxxx
80H	7FFH	110xxxxx 10xxxxxx
800H	FFFFH	1110xxxx 10xxxxxx 10xxxxxx

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.

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.

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  
dd 'A'  
dq 2 dup (0),0  
t db 'abc'
```

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
```

Ille bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze?

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

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

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

Po wykonaniu programu podaj zawartość zmiennej save.

1. mov esi, 0
2. mov ax, 1
3. mov dx, -1
4. cmp ax, dx
5. ja ptc
6. mov esi, 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. 0FEEEDh) 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	01
6000001A	42
60000019	00
60000018	61
60000017	01
60000016	42
60000015	.
	.

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 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 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 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		0xxxxxx	110xxxxx	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

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

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 $x \oplus x = 0$.

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

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

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

```
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
```

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.

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

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).

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 tekst
    lea   eax, [tekst+3]
    push eax
    push 0
    call _MessageBoxA@16
```

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
```

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

```
mov ecx, -1  
mov cx, 22  
skok:  
    sub cx, 11  
    je et2  
    call skok  
et2:  
    mov ecx, [esp]  
    neg ecx  
    lea ecx, [ecx+et2+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.

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

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

```
mov ebx,0ffffFFFFFFh
mov bx,2
etykieta:
    sub bx,1
    jz dalej
    call etykieta
dalej:
    lea ebx,[dalej]
    sub ebx,[esp]
```

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.

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

- - - - - - - -

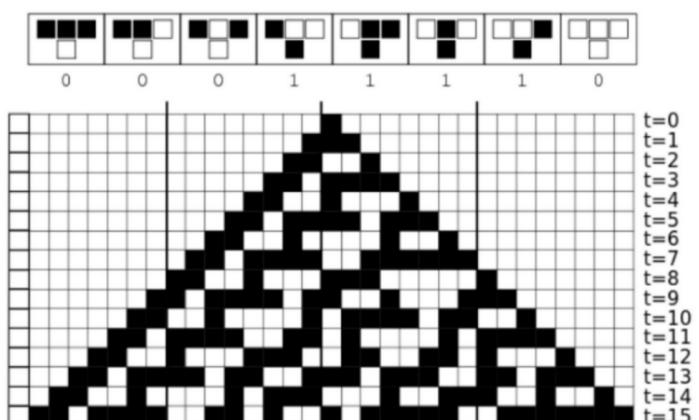
mov ebx, OFFSET pqr
lea edi, [ebx+1]
mov eax, [edi]

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:

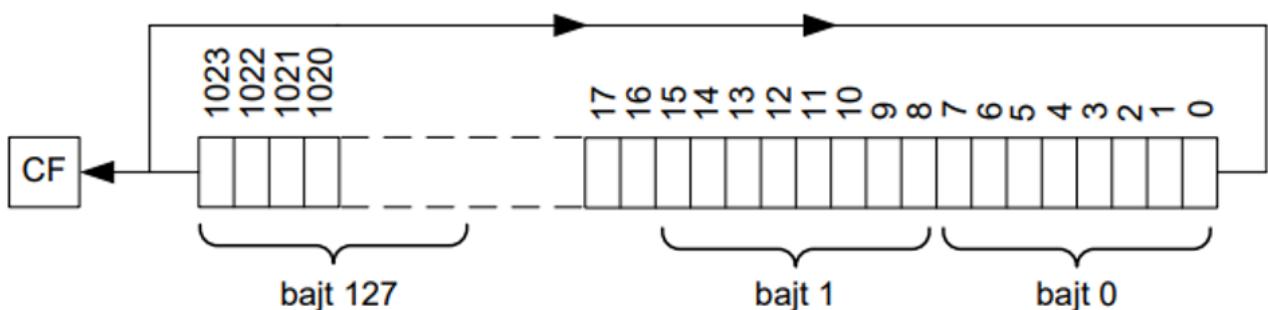
```
starting_row db 0h, 0h, 80h, 0h
next_rows db 15*4 dup(?)
```

Kod powinien sekwencyjnie uzupełnić obszar next_rows właściwymi wartościami.

rule 30



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.

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:

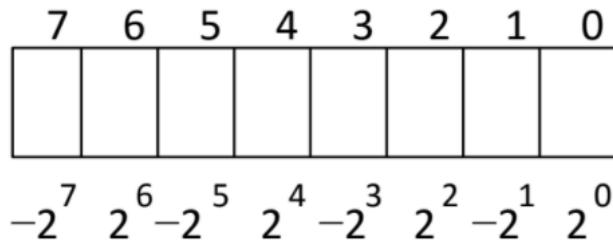
$$\text{dzielna} = \text{iloraz} * \text{dzielnik} + \text{reszta}$$

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).

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.

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.

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

shl eax, 0Bh

sub eax, 4000 0000h



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*.

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 rejestru AH jest większa od 4, jeśli tak, to do rejestrów EAX dodać liczbę 300H.
- c. Zbadać czy liczba umieszczona na bitach 7 – 4 rejestru 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 znajdująca 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.

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

Uwagi:

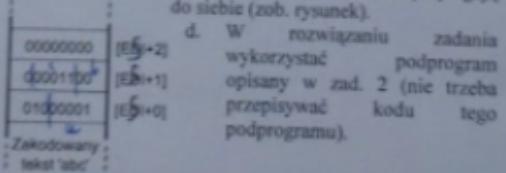
- a. Zapisy $\{ESI\}$ i $\{EDI\}$ oznaczają, odpowiednio, wartości liczb wskazywanych przez rejesty ESI i EDI.
- b. Liczby mogą mieć niejednakową liczbę cyfr.
- c. Operację porównania przeprowadzić bez konwersji obu liczb na postać binarną.

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)

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

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



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

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

