MIARA ŁUKOWA KĄTA

```
.data
liczba dd 180.0
wyswietlacz db 12 dup('?'),10
tysiac dd 1000.0
x1 dd ?
x2 dd ?

.code
_main:
mov ebx,0                       ;nasz kat
mov ecx,5
finit

liczenie:
        mov x1,ebx              ;wpisanie kata do pamieci
        fild x1                 ;zaladowanie kata na wierzcholek stosu koprocesora
        fldpi                   ;zaladowanie liczby pi na wierzcholek stosu koprocesora
        fmulp ST(1),ST(0)       ;kat*pi
        fld liczba              ;zaladowanie liczby 180
        fdivp ST(1),ST(0)       ;(kat*pi)/180    -> obliczenie miary lukowej kata
        FPTAN                   ;obliczenie tangensa (rozkac FDIV po FPTAN jest wymagany)
        FDIV
        fld tysiac              ;zaladowanie liczby 1000
        fmulp ST(1),ST(0)       ;pomnozenie wyniku (obliczonego tangensa) *1000

        fist x2                 ;zaladowanie wyniku do pamieci
        mov eax,x2              ;wprowadzenie wyniku do eax
        call _wyswietl32        ;funkcja ktora wyswietli wynik
        add ebx,0FH             ;dodanie 15 do kata w celu policzenia nastepnego
        loop liczenie
```

ZMIANA PODSTAWY LOGARYTMU + LOGARYTM O PODSTAWIE 2

```
.data
wyswietlacz db 12 dup(?),10
liczba dd ?
tysiac dd 1000

.code
_main:

mov ebx,1                       ;liczenie przedzialu (1-10)
finit
liczenie:
        cmp ebx,11              ;sprawdzenie czy nie jestesmy poza przedzialem
        je zakoncz
        mov liczba,ebx          ;zaladiwanie biezacego argumentu logarytmu do pamieci
        fld1                    ;zaladowanie jedynki na stos (wyjasnienie za chwile, nazwijmy ta jedynke 'q')
        fild liczba             ;zaladowanie argumentu z pamieci na stos (nazwijmy go 'p')
        FYL2X                   ;policzenie logarytm (q* log[podst=2] (p)   -> po to byla jedynka, q w ST(1), p w ST(0))
        fild tysiac             ;zaladowanie liczby 1000 na stos (bedzie w ST(1), poprzedni rozkaz usunac wszystko poza wierzcholkiem)
        fmulp ST(1),ST(0)       ;wynik= wynik *1000 (by wyswietlic do 3 miejsc po przecinku -> do rejestrow tylko calkowite wchodza
                                ;a przy wyswietlaniu tylko kropke trzeba postawic w odpowiednim miejscu)
        fistp liczba            ;zaladowanie czesci calkowitej wyniku do pamieci (z usunieciem ze stosu)
        mov eax,liczba          ;zaladowanie tego samego do eax
        call _wyswietl32        ;wyswietlenie wyniku
        inc ebx
        jmp liczenie
```

OBLICZANIEe^x.data

```
wyswietlacz db 12 dup(?),10
tysiac dd 1000.0
dziesiec dd 10
liczba1 dd ?
liczba2 dd ?

.code
_main:
finit
fld1                                    ;zaladowanie jedynki
fild dziesiec                           ;zaladowanie dziesiatki
FYL2X                                   ;obliczenie logarytmu o podstawie dwa z dziesieciu
;powyzsze bylo potrzebne by zamienic podstawy logarytmu, koprocesor moze
;bezposrednio liczyc tylko logarytm o podstawie 2
;uzylem wiec wzoru na zamiane podstawy logarytmu -> log[podst=10] (x) = [log[podst=2] (x)]/[log[podst=2] (10)]
; i tutaj obliczylem wartosc mianownika
fstp liczba2
mov ebx,1     ;poczatek przedzialu

liczenie:
        cmp ebx,11                      ;sprawdzenie czy nie wyszlismy poza przedzial
        je zakoncz
        mov liczba1,ebx                 ;wprowadzenie obecnego agumentu do tablicy
        fld1                            ;zaladowanie jedynki (wyjasnienie w zadaniu 5-2)
        fild liczba1                    ;zaladowanie argumentu
        FYL2X                           ;policzenie logarytmu o podstawie 2 dla tego argumentu
        fld liczba2                     ;zaladowanie mianownika
        fdivp ST(1),ST(0)               ;licznik/mianownik
        fld tysiac                      ;zaladowanie liczby 1000
        fmulp ST(1),ST(0)               ;pomnozenie wyniku * 1000
        fistp liczba1                   ;zaladowanie do pamieci wraz z usunieciem ze stosu(i zamiana na liczbe calkowita)
        mov eax,liczba1                 ;wprowadzenie wyniku do eax
        call _wyswietl32                ;wyswietlenie wyniku
        inc ebx
        jmp liczenie
```

OBLICZANIE ŚREDNIEJ ARYTMETYCZNEJ

```
int n;
float tablica[15],wynik;
srednia_arytm(tablica,n);

.data
liczba dd ?

.code
_srednia_arytm PROC
mov ecx,[ebp+12]                ;liczba n w ecx
push ecx                        ;zachowanie liczby n na stosie
mov ebx,[ebp+8]                 ;adres tablicy float w ebx
finit
fldz                            ;zaladowanie liczby 0 na stos koprocesora (dotychczasowy wynik)
obliczenia:
        fld dword PTR [ebx]     ;wprowadzenie liczby z tablicy float na stos
        faddp ST(1),ST(0)       ;dodanie do aktualnej zawartosci stosu
        add ebx,4               ;przejscie na nastepna liczbe
        loop obliczenia

        pop ecx                 ;przywrocenie n
        mov liczba,ecx          ;zaladowanie n do pamieci
        fild liczba             ;wprowadzenie liczby n na stos koprocesora
        fdivp ST(1),ST(0)       ;wynik dodawania liczb z tablic podzielony przez n
ret
_srednia_arytm ENDP
```

OBLICZANIE SZEREGU NAPRZEMIENNEGO (parametr n)

```
.data
liczba dd ?
wynik dd ?

.code
_szereg_przemienny PROC
mov ecx, [ebp+8]
mov ebx,2   ;mianownik
finit
fld1                    ;pierwszy wyraz szeregu (1)
dec ecx
obliczenia:
        cmp ecx,0
        je koniec
        mov liczba,ebx          ;wprowadzenie mianownika do pamieci
        fld1                    ;zaladowanie jedynki na stos
        fild liczba             ;zaladowanie mianownika na stos
        fdivp ST(1),ST(0)       ;1/mianownik
        fsubp ST(1),ST(0)       ;(dotychczasowy wynik szeregu) - (1/mianownik)
        shl ebx,1               ;mianownik*2
        dec ecx

        cmp ecx,0
        je koniec
        mov liczba,ebx          ;analogicznie jak wyzej ale dodawanie zamiast odejmowania
        fld1
        fild liczba
        fdivp ST(1),ST(0)
        faddp ST(1),ST(0)
        shl ebx,1
        dec ecx
        jmp obliczenia
koniec:
        ret
_szereg_przemienny ENDP
```

SREDNIA KWADRATOWA

```
        srednia_kwadr(tablica,5);
.data
        liczba dd ?

.code
_srednia_kwadr PROC
        mov ecx,[esp+8]         ;liczba n w ecx
        mov ebx,[edp+4]         ;adres tablicy float w ebx
        push ecx                ;zachowanie n na stosie
        finit
        fldz                    ;zaladowanie zera na wierzcholek stosu koprocesora

        ;najpierw liczymy mianownik
        obliczenia:
                fld dword PTR [ebx]     ;pobranie kolejnej liczby
                fmul ST(0),ST(0)        ;podniesienie jej do kwadratu
                faddp ST(1),ST(0)       ;dodanie do biezacego wyniku
                add ebx,4               ;przejscie na kolejny element tablicy
                loop obliczenia

                pop ecx                 ;przywrocenie liczby n
                mov liczba,ecx          ;zapisanie jej w pamieci
                fild liczba             ;przepisanie jej z pamieci na stos koprocesora
                fdivp ST(1),ST(0)       ;podzielenie licznika przez n
                fsqrt                   ;spierwiastkowanie wyniku
        ret
_srednia_kwadr ENDP
```

ŚREDNIA HARMONICZNA

```
                srednia_harm(tablica,5);

.data
n dd ?

.code
_srednia_harm PROC
        mov ecx,[esp+8]                 ;liczba n w ecx
        mov n,ecx                       ;liczba n zapamietana w pamieci
        mov ebx,[esp+4]                 ;adres tablicy float w ebx
        finit
        fild n                          ;zaladowanie n na wierzcholek stosu koprocesora
        ;najpierw policzymy mianownik
        fldz                            ;zaladowanie zera na stos koprocesora (aktualna wartosc mianownika)
        obliczenia:
                fld1                    ;zaladowanie jedynki na stos koprocesora
                fld dword PTR [ebx]     ;zaladowanie liczby z tablicy float
                fdivp ST(1),ST(0)       ;podzielenie 1/liczba_z_tablicy
```

```
                faddp ST(1),ST(0)               ;dodanie wyniku do mianownik
                add ebx,4                       ;przejscie na nastepna liczbe
                loop obliczenia

        fdivp ST(1),ST(0)                       ;n/mianownik
        ret
_srednia_harm ENDP
```

ALGORYTM SZYBKIEGO PIERWIASTKOWANIA

szybki_pierw(tablica,160);

```
.data

.code
_szybki_pierw PROC
        mov ebx,[esp+4]         ;adres tablicy w ebx
        mov eax,[esp+8]         ;liczba elementow tablicy (n) w eax
                                ;trzeba ja podzielic przez 4 gdyz bedziemy pierwiastkowac
                                ;po 4 liczby na raz, wiec obiegow petli nie bedzie np 160 tylko 40
        mov edx,0
        mov esi,4
        div esi
        mov ecx,eax
        obliczenia:
                movups xmm6,[ebx]               ;zaladowanie czterech liczb do xmm6
                sqrtps xmm5,xmm6                ;spierwiastkowanie ich
                movups [ebx],xmm5               ;zaladowanie czterech liczb do tablicy
                add ebx,16                              ;przejscie w tablicy 4 elementy dalej
                loop obliczenia
        ret
_szybki_pierw ENDP
```

XMM:DODAWANIE/ODEJMOWANIE 1 CO DO DRUGIEGO ELEMENTU TABLICY

pm_jeden(tablica);

```
.data
jedynki dd 4 dup (-1.0)
.code
_pm_jeden PROC
        mov ebx,[esp+4]         ;adres tablicy w ebx
        movups xmm0,[ebx]       ;zaladowanie czterech liczb do xmm0
        movups xmm1,jedynki     ;zaladowanie jedynek do xmm1
        addsubps xmm0,xmm1      ;dodanie/odjecie na odpowiednich pozycjach
        movups [ebx],xmm0       ;przeslanie wyniku do tablicy
        ret
_pm_jeden ENDP
```

XMM WYZNACZANIE MAXIMUM

```
float tablica[]={27.5,143.57,21000.0,-3.51};
float druga[]={27.6,133.57,21070.0,3.51};
wyznacz_max(tablica,druga);
```

```
.code
_wyznacz_max PROC
        mov ebx,[ebp+8]         ;adres pierwszej tablicy w ebx
        mov edi,[ebp+12]        ;adres drugiej tablicy w edi
        movups xmm0,[ebx]       ;zaladowanie pierwszej tablicy do xmm0
        movups xmm1,[edi]       zaladowanie drugiej tablicy do xmm1
        maxps xmm0,xmm1         ;wieksze sposrod pary dwoch liczb wpisywane do xmm0 (wyjasnienie w tresci zadania)
        movups [ebx],xmm0       ;zaladowanie wyniku do pierwszej tablicy
        ret
_wyznacz_max ENDP
```

OBLICZANIE PRZYBLIŻONEJ WARTOŚCI e

nowy_exp(10.0);

```
.data
        argument dd ?
        argument_p dd ?
        wynik dd ?
        mianownik dd ?
        mnoznik dd ?
        testi dd ?
.code
_nowy_exp PROC
        mov eax,[ebp+8]                 ;argument sinh
        mov argument,eax
        mov argument_p,eax
        mov ecx,19                              ;ilosc obiegow petli
        finit
        fld1                                    ;zaladowanie jedynki (pierwszy wyraz szeregu)
        fst wynik
        fst mianownik           ;mianownik (n!)
        fistp mnoznik           ;to przez co musi zostac pomnozony mianownik nastepnym razem
        obliczenia: fld wynik   ;zaladowanie aktualnego wyniku
```

```
                fld argument            ;zaladowanie aktualnego x
                fld mianownik           ;zaladowanie aktualnego mianownika
                fdivp ST(1),ST(0)       ;x/mianownik
                faddp ST(1),ST(0)       ;dotychczasowy wynik + x/mianownik
                fstp wynik              ;zapamietanie wyniku w pamieci
        ;teraz przygotujemy argument i mianownik do nastepnego kroku
                fld mianownik           ;zaladowanie mianownika
                fild mnoznik            ;zaladowanie tego przez co mianownik ma zostac pomnozony
                fld1                            ;zaladowanie jedynki
                faddp ST(1),ST(0)       ;zwiekszenie mnoznika o 1
                fist mnoznik            ;zapamietanie go
                fmulp ST(1),ST(0)       ;mianownik * mnozenik
                fstp mianownik          ;zapamietanie mianownika
                fld argument            ;zaladowanie argumentu
                fld argument_p
                fmulp ST(1),ST(0)       ;podniesienie go do kolejnej potegi
                fstp argument           ;zapamietanie nowo obliczonego argumentu
                loop obliczenia

                fld wynik               ;wpisanie wyniku na wierzcholek stosu koprocesora
        ret
_nowy_exp ENDP
```

NWD

```
_NWD PROC

        mov ebx, [esp + 4]      ;a
        mov ecx, [esp + 8]      ;b
        cmp ebx, ecx
        je rowne

        cmp ebx, ecx
        ja a_wieksze

        jmp b_wieksze

        rowne:
        mov eax, ebx
        ret

        a_wieksze:
        push ecx                ;odlozenie b
        sub ebx, ecx
        push ebx                ;a - b
        call _NWD
        pop ebx
        pop ecx
        ret

        b_wieksze:
        sub ecx, ebx            ;b-a
        push ecx
        push ebx
        call _NWD
        pop ebx
        pop ecx
        ret

  _NWD ENDP
```

SZUKAJ MAX (z liczb x,y,z)

```
_szukaj_max PROC
        push ebp                        ; zapisanie zawartosci EBP na stosie
        mov ebp, esp                    ; kopiowanie zawartosci ESP do EBP
        mov eax, [ebp+8]                ; liczba x
        cmp eax, [ebp+12]               ; porownanie liczb x i y
        jge x_wieksza                           ; skok, gdy x >= y
        ; przypadek x < y
        mov eax, [ebp+12]               ; liczba y
        cmp eax, [ebp+16]               ; porownanie liczb y i z
        jge y_wieksza                           ; skok, gdy y >= z
                                                ; przypadek y < z
                                                ; zatem z jest liczba najwieksza
        wpisz_z: mov eax, [ebp+16]      ; liczba z
        zakoncz:
        pop ebp
        ret
        x_wieksza:
        cmp eax, [ebp+16]               ; porownanie x i z
        jge zakoncz                     ; skok, gdy x >= z
        jmp wpisz_z
        y_wieksza:
        mov eax, [ebp+12]               ; liczba y
        jmp zakoncz
_szukaj_max ENDP
```

64 bit + rekurencja + kopro (1.2 – funkcja(n-1))/n

1 dla n = 1,
2 dla n = 2,
(1.2 – funkcja(n-1))/n dla n > 2,

```
.data
jedendwa dd 1.2

.code
oblicz PROC
        push rbp
        mov rbp, rsp

         mov [rbp + 16], rcx ; Kopia rcx do shadowspace
        cmp rcx, 2
        ja dalej
        fild qword PTR [rbp + 16]
        jmp wypisz_kop
        dalej:


        ; Oblicz(n - 1)
        dec rcx
        sub rsp, 32
        call oblicz
        add rsp, 32

        movss dword PTR [rsp + 24], xmm0
        fild qword PTR [rbp + 16]
        fld dword PTR jedendwa
        fld dword PTR [rsp + 24]
        fsubp
        fdivrp

        wypisz_kop:
        fstp dword PTR [rbp + 16]
        movss xmm0, dword PTR [rsp + 16]        ; movss(sd) - mov single(double)-precision floating-number value

        pop rbp
        ret
oblicz ENDP
```

MNOŻENIE EAX * 100 (PRZESUWANIE REJESTRU)

a × 100 = a × 64 + a × 32 + a × 4.

```
        mov ebx, eax        ; ebx = a
        mov ecx, eax        ; ecx = a
```

```
        sal eax, 2          ; eax = a*4
        sal ecx, 5          ; ecx = a*32
        sal ebx, 6          ; ebx = a*64
        add eax, ecx        ; eax = a*4 + a*32
        add eax, ebx        ; eax = a*4 + a*32 + a*64
```

OBLICZANIE KWADRATU Z LABOREK

a2 = (a – 2)2 + 4*a – 4 dla a > 1
a2 = 1 dla a = 1
a2 = 0 dla a = 0

```
_kwadrat PROC
        push ebp
        mov ebp,esp
        mov eax,[ebp+8]  ;wrzucam a do eax
        cmp eax,0  ; jak a=0 to zwroc wynik
        je koniec
        cmp eax,1  ; jak a=1 to zwroc wynik
        je koniec
        mov ecx,eax         ; 4*a
        add ecx,eax
        add ecx,eax
        add ecx,eax
        sub ecx,4   ; 4*a - 4
        sub eax,2   ; a-2
        push eax
        call _kwadrat       ;rekurencja
        add esp,4   ;balansowanie stosu
        add eax,ecx         ; (a-2)^2 w eax, 4*a-4 w ecx
koniec: pop ebp
        ret
ENDP
```

SZUKA PIERWSZEGO WYSTAPIENIA PODANEGO ZNAKU W PODANYM TEKSCIE (zwraca adres znaku)

public _strstr ; char *strstr(const char *string, const char *strCharSet)

```
.code
_strstr:
push ebp
mov ebp, esp
push edi, esi, ebx

;inicjalizacja
mov esi, [ebp+8]        ;*string, będzie inkrementowane
mov edi, [ebp+12]       ;*strCharset, używamy modyfikatora

mov eax, esi            ;wyjście, przesuwane jeśli nie pasuje
mov ebx, 0              ; modyfikacja adresowac strCharset

;iterowanie
_ptla:
        mov dl, [esi]
        mov cl, [edi+ebx]
        cmp cl, 0
        je _znaleziono
        cmp dl, 0
        je _nieznaleziono
        cmp dl, cl
        je _rowne           ; mamy rowne znaki
        ;jak nierowne to rob:
        mov ebx, -1         ; bo incujemy, unikamy zbednych skokow
        mov esi, eax
        inc eax             ;uzywamy indeksu o jeden wiekszego niz pozycja wykrytego poprzednio podciagu
                            ;poniewaz, np. moze byc ccd a chcemy znalesc cd
        _rowne:
                inc esi
                inc ebx
jmp _ptla

_nieznaleziono:
mov eax, esi

_znaleziono:
pop ebx, esi, edi, ebp
ret
```

SUMOWANIE TABLIC (SSE)

```
; Program przykładowy ilustrujacy operacje SSE procesora
; Poniszy podprogram jest przystosowany do wywoływania
; z poziomu jezyka C (program arytmc_SSE.c)
.686
.XMM ; zezwolenie na asemblacje rozkazów grupy SSE
.model flat
        public _dodaj_SSE, _pierwiastek_SSE, _odwrotnosc_SSE
```

```
.code
_dodaj_SSE PROC
        push ebp
        mov ebp, esp
        push ebx, edi, esi

        mov esi, [ebp+8]                        ; adres pierwszej tablicy
        mov edi, [ebp+12]                       ; adres drugiej tablicy
        mov ebx, [ebp+16]                       ; adres tablicy wynikowej
        ; ładowanie do rejestru xmm5 czterech liczb zmiennoprzecin-kowych 32-bitowych - liczby zostaja pobrane z tablicy, której adres poczatkowy
        podany jest w rejestrze ESI
        ; interpretacja mnemonika "movups" : mov - operacja przesłania,  u - unaligned (adres obszaru nie jest podzielny przez 16), p - packed (do
        rejestru ładowane sa od razu cztery liczby), s - short (inaczej float, liczby zmiennoprzecinkowe 32-bitowe)

        movups xmm5, [esi]
        movups xmm6, [edi]

        addps xmm5, xmm6                ; sumowanie czterech liczb zmiennoprzecinkowych zawartych w rejestrach xmm5 i xmm6

        movups [ebx], xmm5              ; zapisanie wyniku sumowania w tablicy w pamieci
        pop edi, esi, ebx, ebp
        ret
_dodaj_SSE ENDP
```

WYSZUKIWANIE PODCIĄGU W CIĄGU

```
_strstr PROC
        push EBP
        mov EBP, ESP
        push ESI, EDI, EDX,ECX,EBX
        mov ESI, [EBP+8]                ; ciąg
        mov EDI, [EBP+12]               ; podciąg wyszukiwany
        mov ECX, -1                     ; pozycja w ciągu
        mov EDX, -1                     ; pozycja w podciągu
        petla:
                inc EDX
                mov AL, byte PTR [EDI][EDX]
                cmp AL, 0
                je sukces
                petla2:
                inc ECX
                cmp [ESI][ECX], AL
                je petla
        mov EDX, -1
        cmp byte PTR [ESI][ECX], 0
        je porazka
        jmp petla
        sukces:
        sub ECX, EDX
        inc ECX
        porazka:
        lea EAX, [ESI+ECX]
        pop EBX, ECX, EDX, EDI, ESI, EBP

        ret
_strstr ENDP
```

INT - > FLOAT (XMM)

```
_int2float PROC
            push ebp
            mov  ebp, esp
            pusha

            mov esi, [ebp+8]    ; input
            mov edi, [ebp+12]   ; output
            cvtpi2ps xmm5, qword PTR [esi]
            movups [edi], xmm5

            popa
            pop ebp
            ret
_int2float ENDP
```

obliczanie odwrotności czterech liczb zmiennoprzecinkowych (xmm)

```
_odwrotnosc_SSE PROC
        push ebp
        mov ebp, esp
        push ebx
        push esi
        mov esi, [ebp+8]                                        ; adres pierwszej tablicy
        mov ebx, [ebp+12]                                       ; adres tablicy wynikowej
        ; ladowanie do rejestru xmm5 czterech liczb zmiennoprzecinkowych 32-bitowych - liczby zostaja pobrane z tablicy,
        ; której adres poczatkowy podany jest w rejestrze ESI  mnemonik "movups": zob. komentarz podany w funkcji dodaj_SSE
        movups xmm5, [esi]

        rcpps xmm5, xmm6            ; - wynik wpisywany jest do xmm5
        movups [ebx], xmm5          ; zapisanie wyniku sumowania w tablicy w pamieci
        pop esi, ebx, ebp
        ret
_odwrotnosc_SSE ENDP
```

LICZNIK WYSTĄPIEŃ ZNAKÓW W TEKŚCIE

void *wystapienia(void *obszar, unsigned int n);

Funkcja wystapienia powinna utworzyć tablicę wystąpień znaków w 'obszar'.;Rozmiar tego obszaru wynosi n i jest podany jako drugi parametr. Obszar;źródłowy składa się z bajtów o wartościach od 0 do 255. Każdy wiersz tablicy;wystąpień ma składać się z 8-bitowego znaku wejściowego i 32-bitowego;licznika wystąpień. Funkcja powinna zwrócić wskaźnik na utworzoną tablicę wystąpień.;Znaki o zerowej liczbie wystąpień również umieszczone są w tablicy wystąpień.

```
_wystapienia PROC
        push ebp
        mov ebp, esp
        mov esi, dword ptr [ebp+8]          ;obszar
        mov ecx, dword ptr [ebp+12]         ;liczba elementów

        mov eax, 5                          ;na każdy znak będziemy potrzebować 5 bitów
        mul ecx                             ;obliczam, ile łącznie potrzeba bitów
        push ecx                            ;malloc zmienia zawartość ecx, więc robię kopię
        push eax                            ;argument malloc
        call _malloc
        add esp,4                           ;zdejmuję argument ze stosu
        pop ecx
        cmp eax, 0
        je koniec                           ;jeżeli rezerwacja pamięci nie udała się, kończę program

        mov edx,0                           ;jakiś licznik :v
        mov edi,eax;
        push eax                            ;kopia adresu docelowego
        mov eax,0
        ptl:
                mov byte ptr [edi], dl      ;wpisuję do pamięci znak
                mov dword ptr [edi+1], eax  ;zeruję liczbę wystąpień dla znaku
                add edi,5                   ;dane na temat kolejnego znaku będą w pamięci 5 bitów dalej
                inc edx                     ;licznik pętli i jednocześnie numer znaku do wpisania
                cmp edx, 256
                jne ptl

        pop eax                             ;wrzucam do eax adres zarezerwowanej pamięci
        mov edi, eax
        mov ebx,0

        ptl1:
                mov dl, byte ptr [esi]      ;do dl wkładam znak, którego licznik wystąpień muszę zwiększyć
                push edi                    ;backup adresu docelowego
                ptl2:
                ;znaku, który mam w dl, szukam w tablicy wystąpień
                        cmp dl, byte ptr [edi]
                        je znalazlem
                        add edi, 5
                        jmp ptl2
                znalazlem:
                        inc dword ptr [edi+1]   ;pod [edi] kryje się znak, na czterech kolejnych bitach jest licznik wystąpień
```

```
		pop edi			;do edi znów wkładam adres tablicy wystąpień
		inc esi			;przechodzę do kolejnego znaku z badanej tablicy
		inc ebx
		cmp ebx, ecx
		jne ptl1
	koniec:
	pop ebp
	ret
_wystapienia ENDP
```

ZMODYFIKOWANE SORTOWANIE BĄBELKOWE

```
	push ebp
	mov ebp, esp
	push esi, edi, ecx, eax, ebx
	mov esi, [ebp + 8]; tab źródłowa
	mov ecx, [ebp + 12]; ilość el
	dec ecx
	petla1:
		mov ebx, [ebp + 12];n
		dec ebx
		mov eax, byte ptr 8
		mul ebx
		mov ebx, eax
		add ebx, 4

		mov eax, dword ptr[esi + ebx]		; wartość 1 el.tablicy(zaczynamy od końca)
		push ebx				; zapisanie adresu 1 el. na stosie
		petla2:
			sub ebx, 8				; adres 2 el.w ebx
			mov edx, [esi + ebx]			; wartość 2 el.w edx
			cmp eax, edx				; porównanie liczników wystąpień 1 i 2 elementu
			ja zamien				; zamiana jeżeli 1 wiekszy(sortowanie malejace)
			;jeżeli nie doszło do zamiany, również trzeba zaktualizować element, który teraz będzie porównywany
			mov eax, edx				; aktualizacja elementu 1
			add esp, 4				; ściągnięcie ze stosu poprzedniego adresu
			push ebx				; aktualizacja adresu elementu 1
			dalej:
			cmp ebx, 4				; warunek zakończenia petla2
			jne petla2
		pop ebx					; oczyszczenie stosu
		loop petla1				; wyznaczamy kolejny 1 element.
	jmp koniec					; chyba, że koniec pętli(ecx=0)
	zamien:
	mov edi, ebx					; zapis adresu 2 el. do edi
	pop ebx						; w ebx adres 1 el.

	mov[esi + edi], eax;		;zamiana liczników wystąpień
	mov[esi + ebx], edx;

	push eax			; zapis rejestrów na stosie
	push edx

	mov eax, [esi + ebx – 4]	; zamiana bajtów znaku
	mov edx, [esi + edi - 4]
	mov[esi + edi - 4], eax
	mov[esi + ebx - 4], edx

	pop edx				;odtworzenie rejestrów
	pop eax
	mov ebx, edi			; aktualizacja adresu el. 1
	push ebx
	jmp dalej
	koniec:
	pop ebx, eax, ecx, edi, esi, ebp
	ret
```

LICZBA PI

```
.data
	dwa dd 2.0
.code
_liczba_pi PROC
	push ebp
	mov ebp, esp
	push eax
```

```
        push ebx
        mov ecx, [ebp + 8]          ; licznik pętli
        fld dword ptr dwa           ; liczba 2
        fldz; ułamek
        fld dword ptr dwa           ; licznik
        fld1                        ; mianownik
        fld dword ptr dwa           ; wynik
        mov eax, 0                  ; flaga wyznaczająca moment zwiększenia licznika
        mov ebx, 1                  ; flaga wyznaczająca moment zwiększenia mianownika
        pi:
                inc eax
                inc ebx
                ;wyznaczenie ułamka
                fxch st(3);
                fsub st(0), st(0)   ;reset ułamka
                fadd st(0),st(2)    ; licznik
                fdiv st(0), st(1)   ; ułamek w st(0)
                fxch st(3)          ; aktualizacja rejestrów
                fmul st(0), st(3)   ; wynik *= ułamek
                cmp eax, 2;
                je zwieksz_licznik
                cmp ebx, 2
                je zwieksz_mianownik
                loop pi
        koniec:

        pop ebx,eax,ebp
        ret
        zwieksz_licznik:
        mov eax, 0
        fxch st(2)
        fadd st(0), st(4)
        fxch st(2)
        loop pi
        jmp koniec
        zwieksz_mianownik :
                mov ebx, 0
                fxch st(1)
                fadd st(0), st(4)
                fxch st(1)
                loop pi
                jmp koniec
_liczba_pi ENDP
```

ZNAJDŹ MIN

int f(int *tab, int n);

```
_f PROC
        mov ebx, [esp+4]                ;tablica
        mov edi, [esp+8]                ;n elementów

        mov eax, dword ptr [ebx]        ;eax -> min, (domyślnie pierwszy element)
        mov ecx, edi                    ;ecx -> licznik pętli

        znajdz_min:
                cmp eax, [ebx+4*ecx]    ;porównanie
                jb dalej
                mov eax, [ebx+4*ecx]
                dalej:
                loop znajdz_min
        ret
_f ENDP
```

SORTOWANIE TABLICY

int f(int *tab, int n);

```
_f PROC
        mov ebx, [esp+4]                        ;tablica
        mov edi, [esp+8]                        ;n elementów
        mov ecx, 0

        petla:
                mov esi, ecx
                lea eax, [ebx+4*esi]            ;eax wskazuje na min
                znajdz_min:
                        mov edx, [eax]          ;edx -> temp
                        cmp edx, [ebx+4*esi]    ;porównanie
                        jb dalej

                        lea eax, [ebx+4*esi]

                        dalej:
                        add esi, 1
                        cmp esi, edi
                        jne znajdz_min
```

```
                mov edx, [ebx+4*ecx]  ;słaby swap
                mov esi, [eax]
                mov [ebx+4*ecx], esi
                mov [eax], edx

                add ecx, 1
                cmp ecx, edi
                jne petla
ret
```

KOMPRESJA TEKSTU

```
.code
_kompresja_tekstu PROC
        push ebp                ; zapisanie zawartosci EBP na stosie
        mov ebp, esp            ; kopiowanie zawartosci ESP do EBP
        push esi
        push edi
        push ebx
        push ecx
        mov esi, [ebp + 8]      ; tablica źródłowa
        mov edi, [ebp + 12]     ; tablica docelowa
        mov ebx, 0              ; index w tablicy docelowej
        mov ecx, 0              ;licznik pojedyńczego łańcucha
        kompresja:
        mov dh, byte ptr[esi]   ; poprzedni znak(esi jest aktualizowane po zakończeniu łańcucha)
        mov ecx, 0              ; licznik pojdyńczego łańcucha
        cmp dh, 0               ; koniec tablicy
        je koniec_kompresji
        zliczaj_te_same:
        inc ecx
        cmp ecx, 127            ; maxymalna długość łańcucha
        je zapisz_lancuch
        mov dl, byte ptr[esi + ecx]         ; następny znak
        cmp dh, dl                          ; czy kolejny znak jest z danego łańcucha
        je zliczaj_te_same
        add esi, ecx                        ;aktualizacja elementu aktualnego
        ; przerwany ciąg znaków
        cmp cl, 1                           ; sprawdzanie długości łańcucha
        je kopiuj_bez_zmian
        zapisz_lancuch:
        or cl, 10000000B                    ; ustawienie 1 na najstarszym bicie
        mov[edi + ebx], byte ptr dh         ; odesłanie bajtu znaku do docelowej
        mov[edi + ebx + 1], byte ptr cl     ; odesłanie bajtu ilości powtórzeń do docelowej
        add ebx, 2                          ;aktualizacja licznika tablicy docelowej
        jmp kompresja
        koniec_kompresji:
        mov eax, ebx                        ; odesłanie liczby konwertowanych znaków
        pop ecx
        pop ebx
        pop edi
        pop esi
        pop ebp
        ret
        kopiuj_bez_zmian:
        mov[edi+ebx], byte ptr dh
        inc ebx
        jmp kompresja
_kompresja_tekstu ENDP
END
```