



# Kolokwium nr 2 Damian Strojek

## Zadania nr 2 od Pana Dziubicha

1. Poniżej podano fragment programu w języku C.

```
int a, b, * wsk, wynik;
wsk = &b;
a = 21; b = 25;
wynik = roznica(&a, &wsk);
```

Napisać podprogram w assemblerze przystosowany do wywoływanego z poziomu języka C, którego prototyp ma postać:

```
int roznica (int * odjemna,
              int ** odjemnik);
```

Podprogram ten powinien obliczyć różnicę dwóch liczb całkowitych ze znakiem w kodzie U2.

---

## 2. Funkcja biblioteczna języka C o prototype

```
void * malloc(unsigned int k);
```

przydziela k-bajtowy obszar pamięci i zwraca adres przydzielonego obszaru. Jeśli wymagany obszar nie może być przydzielony, to funkcja zwraca wartość 0.  
Napisać podprogram w assemblerze, przystosowany do wywoływania z poziomu języka C — prototyp tego podprogramu ma postać:

```
int * kopia_tablicy(int tabl[],  
                     unsigned int n);
```

Podprogram kopia\_tablicy tworzy nową tablicę o rozmiarach identycznych z oryginalną i zwraca adres nowej tablicy (albo 0, jeśli tablicy nie można było utworzyć).

Do nowej tablicy należy skopiować wszystkie elementy tablicy oryginalnej o wartościach będących liczbami parzystymi. Pozostałe elementy nowej tablicy wypełnić zerami.

*Wskazówki:*

1. Nową tablicę należy utworzyć poprzez przydzielenie odpowiedniego obszaru pamięci za pomocą funkcji malloc.
2. Sprawdzenie czy liczba jest parzysta najłatwiej wykonać poprzez odczytanie najmłodszego bitu liczby (bit nr 0).



3. Podprogram, przystosowany do wywoływania z poziomu języka C, o prototypie:

```
char * komunikat (char * tekst);
```

rezerwuje obszar pamięci (za pomocą funkcji `malloc`) i wpisuje do niego tekst wskazany przez parametr `tekst`, bezpośrednio za którym zostaje dopisany łańcuch znaków "Błąd.". Podprogram zwraca adres przydzielonego obszaru. Napisać kod asemblerowy tego podprogramu.  
Wskazówka: łańcuch znaków kończy się bajtem o wartości 0.

**4.** Poniżej podano fragment programu w języku C.

```
int pomiary[7], * wsk;
-----  
wsk = szukaj_elem_min(pomiary, 7);  
printf("\nElement minimalny = %d\n",  
      * wsk);
```

Napisać podprogram w asemblerze przystosowany do wywoływania z poziomu języka C, którego prototyp ma postać:

```
int * szukaj_elem_min (  
                      int tablica[], int n);
```

Podprogram ten powinien wyznaczyć najmniejszy element tablicy i zwrócić adres (wskaźnik) tego elementu. Liczbę elementów tablicy określa drugi parametr funkcji.



**5.** Podprogram, przystosowany do wywoływania z poziomu języka C, o prototypie:

```
void szyfruj (char * tekst);
```

szyfruje każdy bajt obszaru `tekst` poprzez wykonanie operacji XOR, której drugim argumentem jest ciąg 8 bitów. Ciąg ma inną postać dla każdego bajtu i stanowi 8 najmłodszych bitów 32-bitowej liczby losowej. Pierwsza liczba losowa ma wartość 52525252H, a następne obliczane są w poniższy sposób:

- a. wyznacza się sumę modulo dwa bitów nr 30 i 31,
- b. przesuwa się całą liczbę 32-bitową o jedną pozycję w lewo,
- c. na bit nr 0 liczby wprowadza się wcześniej obliczoną sumę modulo dwa.

Napisać kod podprogramu `szyfruj` w asemblerze.

6. Napisać podprogram w asemblerze obliczający wartość funkcji kwadrat metodą rekurencyjną korzystając z zależności:

$$\begin{aligned}a^2 &= (a - 2)^2 + 4*a - 4 \quad \text{dla } a > 1 \\a^2 &= 1 \quad \text{dla } a = 1 \\a^2 &= 0 \quad \text{dla } a = 0\end{aligned}$$

przy czym argument  $a$  jest liczbą całkowitą 32-bitową zawartą w przedziale  $<1, 65535>$ .

Podprogram powinien być przystosowany do wywoływania z poziomu języka C, a jego prototyp ma postać:

```
unsigned int kwadrat (unsigned int a);
```

W podprogramie nie można używać rozkazów mnożenia i rozkazów przesunięć.



7. W pewnym programie używana jest funkcja iteracja, której prototyp ma postać:

```
unsigned char iteracja(unsigned char a);
```

Podać wartość zwracaną przez funkcję iteracja w poniższym fragmencie programu w języku C

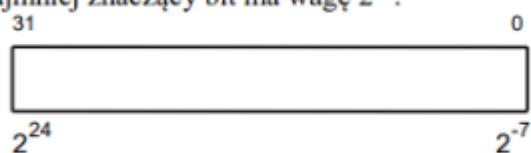
```
w = iteracja(32);
```

Kod funkcji iteracja zapisany w asemblerze ma postać

```
_iteracja      PROC
    push    ebp
    mov     ebp, esp
    mov     al, [ebp+8]
    sal     al, 1
; SAL wykonuje przesunięcie logiczne
; w lewo
    jc     zakoncz
    inc    al
    push   eax
    call   _iteracja
    add    esp, 4
    pop    ebp
    ret

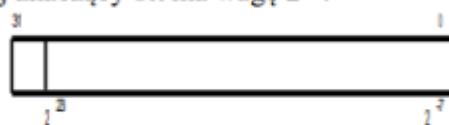
zakoncz:   rcr    al, 1
; rozkaz RCR wykonuje przesunięcie
; cykliczne w prawo przez CF
    pop    ebp
    ret
_iteracja      ENDP
```

- 
8. W programie asemblerowym zdefiniowano format 32-bitowych liczb mieszanych bez znaku, przyjmując, że najmniej znaczący bit ma wagę  $2^{-7}$ .



Napisać fragment programu w assemblerze, który wpisze 1 do znacznika CF (rozkaz STC) jeśli część całkowita liczby zawartej w rejestrze EBX jest różna od zera; w przeciwnym razie CF powinien zostać wyzerowany (rozkaz CLC).

9. W programie zdefiniowano format 32-bitowych liczb mieszanych w kodzie U2 przyjmując, że najmniej znaczący bit ma wagę  $2^{-7}$ .

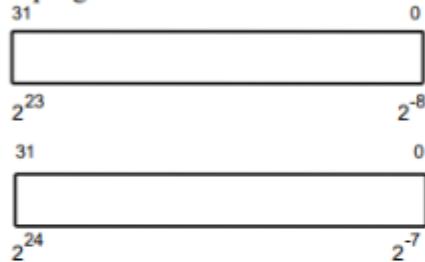


Przyjmując, że liczba w podanym formacie znajduje się w rejestrze EDX, napisać fragment programu, który zaokrągli tę liczbę do najbliższej liczby całkowitej. Wynik należy pozostawić również w rejestrze EDX w podanym formacie.

Wskazówka: zbadać stan bitu o wagie  $2^{-1}$ .



**10.** Na poniższym rysunku pokazane są dwa formaty 32-bitowych liczb staloprzecinkowych bez znaku używane w programie.



Zakładając, że w rejestrze ESI znajduje się liczba zakodowana wg pierwszego formatu (najmniej znaczący bit ma wagę  $2^{-8}$ ), a w rejestrze EDI liczba zakodowana wg drugiego formatu (najmniej znaczący bit ma wagę  $2^{-7}$ ), napisać fragmentu, który porówna obie liczby. Jeśli liczba w rejestrze ESI jest większa od liczby w rejestrze EDI, to do CF należy wpisać 1, w przeciwnym razie 0 (rozkazy STC/CLC).

**11.** Podać reprezentację liczby  $-0.25$  w formacie zmiennoprzecinkowym *double*. Wskazówka: pole mantysy w formacie w formacie *double* zajmuje 52 bity.

---

**12.** Poniżej podano reprezentację binarną dwóch 32-bitowych liczb binarnych zmiennoprzecinkowych (format *float*)  $x$  i  $y$ . Ile wynosi iloraz  $x/y$  tych liczb ?

|   |          |                                  |
|---|----------|----------------------------------|
| 0 | 10000001 | 00000000000000000000000000000000 |
| 1 | 10000000 | 00000000000000000000000000000000 |

13. W pamięci komputera, począwszy od adresu podanego w rejestrze ESI znajduje się 32-bitowa liczba zmiennoprzecinkowa w formacie *float*. Napisać fragment programu, który przekształci tę liczbę na 64-bitowy format *double* i uzyskany rezultat wpisze do pamięci począwszy od adresu podanego w rejestrze EDI. W omawianym fragmencie nie można używać rozkazów koprocesora arytmetycznego.

14. Poniższy program w języku C wczytuje promień  $r$  koła z klawiatury, oblicza pole koła i wyświetla na ekranie wynik obliczenia.

Obliczenie pola koła wykonuje podprogram zakodowany w asemblerze, przystosowany do wywoływania z poziomu języka C, którego prototyp na poziomie języka C ma postać:

```
void pole_kola (float * pr);
```

Argument  $pr$  jest adresem (wskaźnikiem) zmiennej, w której przechowywany jest promień  $r$  koła..

Napisać podprogram w asemblerze dokonujący opisanego obliczenia i uruchomić program składający się z plików źródłowych w języku C i w asemblerze.

15. Niżej opisany podprogram, przystosowany do wywoływania z poziomu języka C, posiada prototyp:

```
float avg_wd (int n, void * tablica, void  
* wagî)
```

Napisz funkcję avg\_wd w assemblerze z wykorzystaniem rozkazów koprocesora. Zakładając, że tablica i wagî są n-elementowymi tablicami liczb zmiennoprzecinowych typu *float*, należy obliczyć średnią ważoną wszystkich wyrazów tablicy. Przed zakończeniem podprogramu wszelkie wyniki pośrednie znajdujące się na stosie rejestrów koprocesora powinny zostać usunięte.

**16.** Napisać podprogram w asemblerze przystosowany do wywoływania z poziomu języka C, którego prototyp ma postać:

```
unsigned int liczba_procesorow();
```

który zwróci jako wyjście liczbę procesorów w systemie. Napisać przykładowy program w języku C demonstrujący użycie funkcji.

*Wskazówki:*

1. użyj funkcji GetSystemInfo (opis na [msdn.microsoft.com](http://msdn.microsoft.com)).

**17.** Napisać podprogram w asemblerze, przystosowany do wywoływania z poziomu języka C, o prototype:

```
unsigned __int64 sortowanie (unsigned  
__int64 * tabl, unsigned int n);
```

Podprogram powinien posortować zawartość n-elementowej tablicy tabl w porządku rosnącym i zwrócić wartość największego elementu. Elementami tablicy są liczby 64-bitowe bez znaku, a program wykonywany jest w trybie 32-bitowym.

*Wskazówka:* 64-bitowe wartości funkcji w trybie 32-bitowym przekazywane są przez rejesty EDX:EAX.



---

**18.** Napisać podprogram w asemblerze, przystosowany do wywoływania z poziomu języka C, o prototype:

```
wchar_t * ASCII_na_UTF16(char * znaki,  
                           int n);
```

Podprogram powinien przekształcić ciąg n znaków w kodzie ASCII (kody < 128) na odpowiadający mu ciąg znaków w standardzie UTF16. Uzyskane znaki należy wpisać do obszaru pamięci przydzielonego za pomocą funkcji `malloc`. Na końcu ciągu znaków należy umieścić liczbę 0 (16-bitową). Adres przydzielonego obszaru należy zwrócić jako wartość funkcji. Typ `wchar_t` reprezentuje znaki 16-bitowe używane w Unikodzie. Funkcja biblioteczna języka C `malloc(k)` przydziela k-bajtowy obszar pamięci i zwraca adres przydzielonego obszaru. Zawartość przydzielonej pamięci jest nieokreślona.



19. Przedstawić podane rozkazy w postaci ciągów liczb szesnastkowych zapisanych przy użyciu dyrektywy db.

```
Je      dalej  
ppp:    and     eax, 0FH  
        mov     [edx+ebx], dl  
        inc     edx  
        loop   ppp  
dalej:
```

## Zadania z dysku z roku 2020

Podaj wartość zmiennej **wynik** po wykonaniu następującego fragmentu kodu. Wynik podaj w formie dziesiętnej z dokładnością do dwóch miejsc po przecinku (np. 123.45).

C:

```
1 int main()
2 {
3     float a = 1.0 + pow(2.0, -21);
4     float b = 1.125f;
5     float wynik = 0.0f;
6     compute(a, b, &wynik);
7 }
```

ASM:

```
1 compute    PROC
2         push    ebp
3         mov     ebp, esp
4
5         mov     eax, [ebp+8]
6         mov     ebx, [ebp+12]
7         mov     edi, [ebp+16]
8
9         and     eax, 7FFFFFFH
10        shl    eax, 23
11        add     ebx, eax
12
13        mov     [edi], ebx
14
15        pop    ebp
16        ret
17 compute    ENDP
```

Po wykonaniu poniższego fragmentu kodu:

```
1 xor ebx,ebx
2 et1:
3 dw 13 dup (9090h), 9090h
4 db 64h, 0A1h ,30h, 0, 0, 0
5 db 8Ah, 60h, 2, 9Eh, 83h, 0D3h,
6 et2:
    add ebx, et2 - et1
```

zawartość rejestru ebx będzie równa:



Wynik podaj jako liczbę dziesiętną (np. 43).

Zakładając wykonanie następującego fragmentu programu podaj w postaci 32 bitowej liczby szesnastkowej (w formacie 12345678h) zawartość liczby L3 (tzn. DWORD PTR L3), przyjmując zaokrąglanie wyniku poprzez odcięcie nieznaczących bitów. Zakładamy również, że stos koprocesora jest początkowo pusty.

```
1 .data
2 L1 dd 001111110000000000000000000000001b
3 L2 dd 010000000000000000000000000000001b
4 L3 dd 0
5
6 .code
7     fld  L1
8     fadd L2
9     fst  L3
```

Podprogram o następującym prototypie

```
char get_char (unsigned int n)
```

ma zwrócić n-ty znak pewnego łańcucha (numerowany od 0). Łańcuch ten jest tworzony w ciele funkcji get\_char i ulokowany jako zmienna lokalna na stosie. Zakłada się, że jego długość wypełnia całą zaalokowaną wielkość.

Przechowywany jest zgodnie z regułą little endian, od najbliższego adresu.

Uzupełnij brakującą instrukcję (linia 16), w taki sposób, aby w rejestrze AL został zwrócony właściwy element łańcucha. Odpowiedź podaj bez jakichkolwiek spacji w instrukcji.

```
1  get_char      PROC
2      push    ebp
3      mov     ebp, esp
4      sub     esp, 12      ; miejsce na wynikowy łańcuch
5
6      [.... tutaj odłożenie używanych rejestrów]
7
8      mov     eax, [ebp+8]    ; pobranie indeksu znaku c
9
10     [... tutaj znajduje się kod wyznaczający i zapisuje
11         zawartość eax nie ulega w tym miejscu zmianie
12
13
14     zakoncz:
15
16     mov al,   □
17
18     pop    esi
19     add    esp, 12
20     pop    ebp
21     ret
22     get_char    ENDP
23
```



Zakładając wykonanie następującego fragmentu programu podaj w postaci 32 bitowej liczby szesnastkowej (w formacie 12345678h) zawartość liczby L3 (tzn. DWORD PTR L3), przyjmując zaokrąglanie wyniku poprzez odcięcie nieznaczących bitów. Zakładamy również, że stos koprocesora jest początkowo pusty.

```
1 .data
2 L1 dd 01000000000000000000000000000001b
3 L2 dd 01000000100000000000000000000001b
4 L3 dd 0
5
6 .code
7     fld  L1
8     fadd L2
9     fst  L3
```

Podczas komplikacji 32 bitowego programu w asemblerze programista otrzymał następujący błąd:

```
funkcje.asm(17): error A2070: invalid instruction operands
```

Linia 17 w pliku funkcje.asm wygląda następująco:

```
17 pcmpeqb xmm1, 12132h
```

Podaj przyczynę tego błędu.

---

Zakładając wykonanie następującego fragmentu programu podaj w postaci 32 bitowej liczby szesnastkowej (w formacie 12345678h) zawartość liczby L3 (tzn. DWORD PTR L3), przyjmując zaokrąglanie wyniku poprzez odcięcie nieznaczących bitów. Zakładamy również, że stos koprocesora jest początkowo pusty.

```
1 .data
2 L1 dd 01000000010000000000000000000000000000001b
3 L2 dd 01000000010000000000000000000000000000001b
4 L3 dd 0
5
6 .code
7     fld  L1
8     fadd L2
9     fst  L3
```

Po wykonaniu poniższego fragmentu kodu:

```
1     mov ecx,OFFSET et2
2 et1:
3     db 12 dup (90h), 90h
4     db 64h, 0A1h, 30h, 0, 0, 0
5     db 8Ah, 60h, 2, 9Eh, 83h, 0d1h, 0
6 et2:
    sub ecx, OFFSET et1
```

zawartość rejestru ecx będzie równa:



Podaj zawartość rejestru **EDX** w postaci 32-bitowej liczby szesnastkowej (np. DEAD9696H) po wykonaniu poniższego fragmentu kodu, zakładając, że w rejestrze **EDX** i **EBX** znajdują się odpowiednio wartości **edx**: 00000004H, **ebx**: 00000001H.

```
1     stc
2     sbb      edx, edx
3     neg      edx
4     bts      ebx, edx
```

---

Podaj zawartość rejestru **EAX** w postaci 32-bitowej liczby szesnastkowej (np. DEAD9696H) po wykonaniu poniższego fragmentu kodu, zakładając następujące wartości w rejestrach **al: 128, bl: 129**.

|   |            |                 |
|---|------------|-----------------|
| 1 | <b>add</b> | <b>al, bl</b>   |
| 2 | <b>sbb</b> | <b>ebx, ebx</b> |
| 3 | <b>neg</b> | <b>ebx</b>      |
| 4 | <b>bts</b> | <b>eax, ebx</b> |

W programie asemblerowym zdefiniowano format 32-bitowych liczb mieszanych bez znaku, przyjmując, że najmniej znaczący bit ma wagę -2<sup>1</sup>.



Uzupełnij fragment programu w asemblerze, którego celem jest zwiększenie wartości liczby znajdującej się w rejestrze eax o 2. (format liczby hexadecymalny z h i bez jakichkolwiek spacji rozdzielających np. 12345h ).

```
mov eax, 2
```

Poniżej podano reprezentację binarną dwóch 32-bitowych liczb binarnych zmiennoprzecinkowych (format *float*) x i y. Ile wynosi iloraz x/y tych liczb? Wynik podać w postaci zwykłej liczby dziesiętnej (trzy cyfry po kropce).

|   |   |          |                                  |
|---|---|----------|----------------------------------|
| x | 0 | 01111111 | 01000000000000000000000000000000 |
| y | 1 | 10000001 | 10000000000000000000000000000000 |

Podaj zawartość rejestru **EAX** w postaci 32-bitowej liczby szesnastkowej (np. DEAD9696H) po wykonaniu poniższego fragmentu kodu, zakładając następujące wartości w rejestrach **al: 128, bl: 129**.

```
1      add      al, bl
2      sbb      ebx, ebx
3      neg      ebx
4      bts      eax, ebx
```



Poniżej zaprezentowano funkcję wyliczającą pole koła ( $p=\pi \times r^2$ ) o następującym prototypie:

C:

```
1 float compute(int r);
2
3 ...
4 float wynik = compute(a);
5 ...
```

Argumentem funkcji jest zmienna typu INT. Uzupełnić część assemblerową tak aby funkcja zwróciła poprawny wynik.

ASM:

```
1 _compute      PROC
2     push    ebp
3     mov     ebp, esp
4
5     finit
6     [ ]        dword ptr [ebp+8] ; load argument
7
8     fmulp   [ ], ST(0)          ; r^2
9
10    [ ]           ; load pi by no-argument instruction
11
12    fmulp
13
14
15    pop     ebp
16    ret
17 _compute      ENDP
```

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 dw 50 dup (0DEADH)
3. .code
  
4. \_main PROC
5. mov eax, dword PTR OFFSET destination
6. mov [eax + \_\_\_\_], word PTR -1

Ille bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze?

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

---

```
v1 dq ?, ?
      dd 4 dup ('A'), 20
v3 db 10 dup (?)
      dw ('a')
```

## Zadania z dysku z roczników <2020

**12.** Poniżej podano reprezentację binarną liczby  $-8.25$  w postaci 32-bitowej liczby zmiennoprzecinkowej (*float*). Uzupełnić brakujące bity mantisy w tej reprezentacji.

|  |          |       |                          |
|--|----------|-------|--------------------------|
|  | 10000010 | ..... | 000000000000000000000000 |
|--|----------|-------|--------------------------|

**16.** Poniżej podano kod (niepełny) podprogramu w asemblerze, który zwiększa o 1 liczbę zmiennoprzecinkową w formacie *double*. Podprogram przystosowany jest do wywoływania z poziomu języka C, a jego prototyp ma postać:

```
double plus_jeden (double x);
```

Nie używając rozkazów koprocesora, uzupełnić brakujący fragment podprogramu, przy założeniu, że liczba *x* jest większa od 1, a zawartość pola wykładnika (w formacie *double*) należy do przedziału  $<1023, 1075>$ . Uwaga: po wykonaniu dodawania przeprowadzić normalizację liczby.



**18.** Napisać podprogram w asemblerze, przystosowany do wywoływania poziomu języka C, o prototypie:

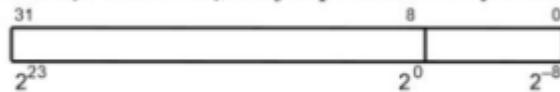
```
unsigned int NWD(unsigned int a,  
                  unsigned int b);
```

Podprogram ten powinien wyznaczyć wartość największego wspólnego podzielnika dla dwóch liczb a i b. Obliczenie należy przeprowadzić metodą rekurencyjną wg algorytmu przedstawionego poniżej.

```
unsigned int NWD(unsigned int a,  
                  unsigned int b)  
{  
    unsigned int wyn;  
    if (a == b) wyn = a;  
    else if (a > b) wyn = NWD (a - b, b);  
    else wyn = NWD (a, b - a);  
    return wyn;  
}
```



**19.** W programie zdefiniowano 32-bitowy typ MIESZ32, w którym kodowane są liczby stałoprzecinkowe bez znaku zawierające część całkowitą i ulamkową, tak jak pokazano na rysunku.



Napisać podprogram w asemblerze przystosowany do wywoływania z poziomu języka C, który zamieni liczbę  $p$  w podanym formacie na 32-bitową liczbę zmiennoprzecinkową w formacie *float*. Prototyp podprogramu na poziomie języka C ma postać:

```
float miesz2float (MIESZ32 p);
```

*Zalecenia i wskazówki:*

- a. Podprogram powinien przeprowadzić konwersję posługując się rozkazami procesora (nie koprocesora). Wartości typu MIESZ32 przekazywane są przez stos wg takich samych reguł jak wartości typu int. Pominąć przypadek liczby 0.
- b. Odszukać położenie najstarszego bitu o wartości 1 w liczbie  $p$ , i następnie przesunąć tę liczbę w prawo lub w lewo o odpowiednią liczbę bitów, dostosowując ją do formatu *float*.
- c. Wyznaczyć także wartość pola wykładnika i wpisać ją na odpowiednie bity. Wynik w postaci liczby typu *float* umieścić na wierzchołku stosu koprocesora.



**21.** Napisać podprogram w asemblerze przystosowany do wywoływania z poziomu języka C, o podanym niżej prototypie

```
float float_razy_float (float a, float b);
```

Podprogram ten powinien wyznaczyć iloczyn dwóch liczb zmiennoprzecinkowych typu float. Obliczenie wykonać bez wykorzystania rozkazów koprocesora arytmetycznego (z wyjątkiem FLD). Zakładamy, że obie liczby są dodatnie, a ich iloczyn da się przedstawić w postaci liczby typu float.

*Wskazówki:*

- a. Zadanie wymaga wyznaczenia iloczynu mantys obu liczb i dodania wykładników (wykładnik jest przesunięty o 127).
- b. W wyniku mnożenia dwóch liczb binarnych, w których najmłodszy bit ma wagę  $2^{-23}$  uzyskujemy iloczyn, w którym najmłodszy bit ma wagę  $2^{-46}$ . Należy dostosować format tego iloczynu do wymagań formatu float.
- c. Jeśli wartość iloczynu jest większa lub równa od 2, to należy go znormalizować i odpowiednio skorygować wykładnik.



|  |          |    |
|--|----------|----|
| 1. (1 pkt.) Zakładając, że OFFSET        | 20200119 | C0 |
| dana=20200116 i zawartość pamięci        | 20200118 | F0 |
| operacyjnej jak na rys. po prawej, podaj | 20200117 | 00 |
| wartość rejestru st(0) po wykonaniu      | 20200116 | 00 |
| rozkazu                                  | 20200115 | 00 |
| FLD DWORD PTR dana                       | 20200114 | 78 |
| Wartość proszę wyrazić w formie liczby   | 20200113 | 41 |
| Odp: .....                               |          |    |

*Imię i nazwisko*

2. (1 pkt) W rejestrze EAX znajduje się liczba zmiennoprzecinkowa w formacie float. Napisać fragment programu w asemblerze, który pomnoży tę liczbę przez 32, przy czym we fragmencie nie mogą występować rozkazy koprocesora arytmetycznego.



3. (4 pkt.) Napisać podprogram w asemblerze, przystosowany do wywoływania z poziomu języka C, o prototypie

```
void aktualna_godzina (char * godzina);
```

Podprogram ten powinienny uzyskać informacje o aktualnej godzinie korzystając z funkcji `GetSystemTime` systemu Windows. Argumentem tej funkcji jest wskaźnik do struktury `SYSTEMTIME`.

```
typedef struct _SYSTEMTIME {  
    WORD wYear;  
    WORD wMonth;  
    WORD wDayOfWeek;  
    WORD wDay;  
    WORD wHour;  
    WORD wMinute;  
    WORD wSecond;  
    WORD wMilliseconds;  
} SYSTEMTIME;
```

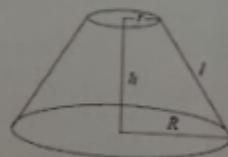
W szczególności pole tego obszaru o nazwie `wHour` zawiera wartość aktualne godziny, którą trzeba przekonwertować do łańcucha znaków i zwrócić jako wynik działania funkcji w formacie 24 godzinnym. W kodzie asemblerowym wywołanie tej funkcji ma postać:

```
call    _ GetSystemTime@4
```

2. (4 pkt.) Napisać podprogram w asemblerze, przystosowany do wywoływania z poziomu języka C, o prototypie:

```
float obj_stozka_sc(float r, float R,  
float h);
```

Podprogram ten powinien  
wyznaczyć objętość  $V$   
stożka ściętego o  
wysokości  $h$ , promieniu  
podstawy dolnej  $R$  i górnej  
 $r$  wg poniższego wzoru:



$$V = \frac{\pi}{3} h(R^2 + Rr + r^2)$$

## Kodowanie rozkazów

Zamień poniższe instrukcje na ciągi liczb reprezentujące rozkazy procesora.

Odpowiedzi należy podać w formie:

8 bitowych liczb szesnastkowych

bez zer przed symbolami

bez znaku H lub h na końcu

z dodatkowymi spacjami pomiędzy bajtami, np. 23 12 AB CD

|                        |                      |   |
|------------------------|----------------------|---|
| et: movsx ebx,bx       | <input type="text"/> | ✓ |
| sub eax,78ABh          | <input type="text"/> | ✓ |
| jg et                  | <input type="text"/> | ✓ |
| add dx,[edi+8*ebx-14h] | <input type="text"/> | ✓ |

```
pppa:    jc      dalej
          sub     bx, 256
          mov     [edx+ebx], dl
          loop   pppa
dalej:   xor     si, bx
```

---

```
et: movsx ebx,bx    |  ✓
sub  eax,78ABh    |  ✓
```

---

```
et: mov ecx,[edi+4*ebx-11h]
    sub eax,5678h
loop et
mul bx
```