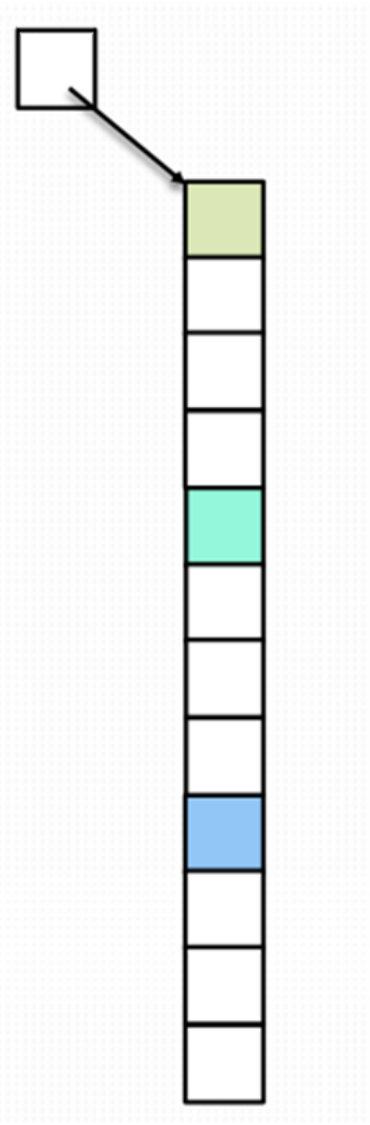
Laboratorium programowania niskopoziomowego

LAB 5 – Operacje na macierzach

O P I S



Macierzą nazywamy tablicę dwuwymiarową o rozmiarach NxM (w szczególnym przypadku NxN). Można rozróżnić tablice dwuwymiarowe statyczne i dynamiczne.

Tablice dwuwymiarowe statyczne są przechowywane w pamięci operacyjnej jako jeden ciąg danych, tak jak tablica jednowymiarowa. Kolejny wiersz jest usytułowany tuż za poprzednim wierszem, bez żadnej przerwy w pamięci operacyjnej. Obrazuje to obrazek po prawej stronie. Poniżej znajduje się kod pozwalający na inicjalizację tablicy i jej późniejsze wypełnienie wartościami pobranymi od użytkownika. W ramach przypomnienia warto zaznaczyć, że zmienne *m* i *n* muszą być zmiennymi stałymi, czyli z kwantyfikatorem *const:*

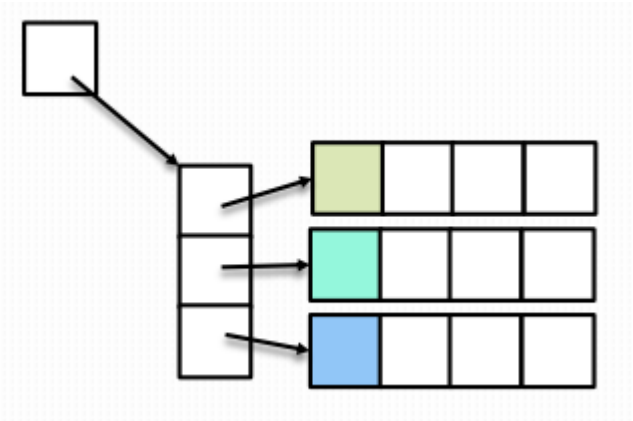
```
#include <iostream>
using namespace std;

const int m = 5, n = 7;
int matrix[m][n];

for (int i = 0; i < m; ++i)
{
	for (int j = 0; j < n; ++j)
	{
		cout << "Podaj: [" << i << "][" << j << "] element tablicy: ";
		cin >> matrix[i][j];
	}
}
```

Oznacza to, że można na dwuwymiarowej tablicy statycznej pracować tak, jak na tablicy jednowymiarowej. Jednakże w przypadkach pracy na wybranych wierszach trzeba zwracać szczególną czujność aby odpowiednio wyliczyć indeks danego wiersza i danej komórki.

W odróżnieniu od tablic dwuwymiarowych statycznych, tablice dwuwymiarowe dynamiczne są inaczej przechowywane w pamięci operacyjnej. Zmienna macierzowa (wskaźnik ***tab*) wskazuje swoim adresem na obszar pamięci, w którym znajduje się jednowymiarowa tablica wskaźników do wierszy (*tab[i]**). Każdy element tej tablicy zawiera wskaźnik na konkretne miejsce w pamięci operacyjnej, gdzie znajduje się jednowymiarowa tablica z danymi odpowiadająca wierszowi macierzy. Bardzo dobrze ilustruje to rysunek znajdujący się po prawej stronie. W ramach przypomnienia znajduje się również przykładowy kod C++ z deklaracją i uzupełnieniem danych w dwuwymiarowej tablicy dynamicznej.



```
#include <iostream>
#include <windows.h>
using namespace std;

	INT64 n = 3;
	INT64 m = 4;
	INT64 **tab = new INT64 *[n];

	for (INT64 i = 0; i < n; ++i)
	{
		tab[i] = new INT64[m];
		for (int j = 0; j < m; ++j)
		{
			cout << "Podaj: " << i << "," << j << " element macierzy: ";
			cin >> tab[i][j];
		}
	}
```

UWAGA – wszystkie tablice utworzone dynamicznie muszą być zwolnione przez programistę.

Wszystkie tablice mogą znajdować się w różnych obszarach pamięci, a to oznacza, że po zakończeniu pracy na jednym wierszu tablicy, aby przejść do innego wiersza, należy pobrać właściwy wskaźnik z tablicy wskaźników.

Druga bardzo ważna informacja: niezależnie od typu danych jaki użytkownik zadeklaruje *INT64 **tab* czy *int**tab* to tablica wskaźników będzie miała ten sam rozmiar. Rozmiar tablicy wskaźników zależy od liczby wierszy oraz od platformy, dla której jest kompilowany kod. Dla:

x86 - wskaźnik ma rozmiar 4 bajtów.
x64 - wskaźnik ma rozmiar 8 bajtów.

Rozmiar tablicy z danymi jest zależny od typu danych jaki użytkownik postanowił zadeklarować, można deklarować tablicę dowolnego typu np. *INT64 (64 bity), short* (16 bitów), czy *byte* (8 bitów).

1. Suma elementów macierzy - x64

1.1. Suma dla macierzy wartości typu int

Przygotuj macierz m x n liczb typu int (32 bitowych). Dołącz: `#include <windows.h>.`
Zadeklaruj funkcję sumy:

```
extern "C" INT64 suma32(int**, int, int);
```

Wpisz funkcję sumy w pliku .asm:

```
.code

suma32 proc ; //RCX = **tab, RDX = m, R8 = n;
	push rsi

	xor rax, rax		;suma=0
petlaM:
	mov rsi, [rcx + 8*rdx - 8] ; // wskaźnik do wiersza
	mov r10, r8 ; n
petlaN:
	movsxd r11, dword ptr[rsi + 4*r10 - 4]
	add rax, r11
	dec r10
	jnz petlaN
	dec rdx
	jnz petlaM

	pop rsi
	ret
suma32 endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj.

1.2. Suma dla macierzy wartości typu INT64

Przygotuj macierz m x n liczb typu INT64. Zadeklaruj funkcję sumy:

```
extern "C" INT64 suma64(INT64**, INT64, INT64);
```

Wpisz funkcję sumy w pliku .asm:

```
.code

suma64 proc uses rsi; //RCX = **tab, RDX = m, R8 = n;

	xor rax, rax		;suma=0
petlaM:
	mov rsi, [rcx + 8*rdx - 8] ; // wskaźnik do wiersza
	mov r10, r8 ; n
petlaN:
	add rax, [rsi + 8*r10 - 8]
	dec r10
	jnz petlaN
	dec rdx
	jnz petlaM

	ret
suma64 endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj. **Zwróć uwagę na deklarację funkcji.**

2. Iloczyn macierzy i wektora - x64

2.1. Iloczyn macierzy i wektora dla wartości typu INT64

Przygotuj macierz m x n liczb typu INT64 oraz wektory o wymiarach n i m. Zadeklaruj funkcję iloczynu:

```
extern "C" INT64 uemxv(INT64**, INT64*, INT64*, INT64, INT64);
```

Wpisz funkcję iloczynu w pliku .asm:

```
.code

uemxv proc uses rbx rsi, mm:ptr, v:ptr, u:ptr, m:qword, n:qword
                    ; //RCX = **mm, RDX = *v, R8 = *u, R9 = m, Stos = n;
       mov r11, n
petlaM:
       mov rsi, [rcx + r9*8 - 8]  ;mm[m]* ustawiamy się na ostatni element
       mov rbx, r11               ;n
       xor r10, r10               ;suma

petlaN:
       mov rax, [rsi + rbx*8 - 8] ; rax = m[m][n]
       imul rax, [rdx + rbx*8 - 8]; rax *= v[n]
       add r10, rax

       dec rbx
       jnz petlaN

       mov [r8 + r9*8 - 8], r10
       dec    r9
       jnz petlaM

       ret
uemxv endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj. **Zwróć uwagę na deklarację funkcji.**

2.2. Iloczyn macierzy i wektora dla wartości typu int

Przygotuj macierz m x n liczb typu int oraz wektory o wymiarach n i m. Zadeklaruj funkcję iloczynu i napisz samodzielnie odpowiedni podprogram.

3. Suma macierzy dwuwymiarowych dla wartości typu INT64

Przygotuj 3 macierze m x n liczb typu INT64. Dołącz: `#include <windows.h>.`
Zadeklaruj funkcję sumy:

```
extern "C" void sumaMeUpV(INT64**, INT64**, INT64**, INT64, INT64);
```

Wpisz funkcję sumy w pliku .asm:

```
.code

sumaMeUpV proc uses rbx rsi rdi, mm:ptr, uu:ptr, vv:ptr, m:qword, n:qword
;RCX = **mm, RDX = **uu, R8 = **vv, R9 = m, Stos = n;
      mov   r11,n

petlaM:
      mov rdi, [rcx + 8*r9 - 8]
      mov rsi, [rdx + 8*r9 - 8]
      mov rbx, [r8 + 8*r9 - 8]
      mov r10, r11

petlaN:
      mov rax, [rsi + 8*r10 - 8]
      add rax, [rbx + 8*r10 - 8]
      mov [rdi + 8*r10 - 8], rax
      dec r10
      jnz petlaN
      dec r9
      jnz petlaM

      ret
sumaMeUpV endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj. **Zwróć uwagę na deklarację funkcji.**
Określ zawartości wszystkich rejestrów.

Napisz funkcję realizującą odejmowanie macierzy postaci: M=M-A.

4. Poszukiwanie elementu minimalnego w macierzy trójwymiarowej

Przygotuj macierz m x n x k liczb typu INT64. Dołącz: `#include <windows.h>.`
Zadeklaruj funkcję minimum:

```
extern "C" INT64 minM3(INT64***, INT64, INT64, INT64);
```

Wpisz funkcję sumy w pliku .asm:

```
.code

minM3 proc uses rsi rdi ; //RCX = ***mmm, RDX = M, R8 = N, R9 = K;

      mov rdi, [rcx]
      mov rsi, [rdi]
      mov rax, [rsi]               ;ustawianie pierwszego elementu

petlaM:
      mov rdi, [rcx + 8*rdx - 8]   ; po i
      mov r10, r8 ; // N
petlaN:
      mov rsi, [rdi + 8*r10 - 8]   ; po j
      mov r11, r9 ; // K
petlaK:
      cmp rax, [rsi + 8*r11 - 8]
                                   ;wersja ze skokiem
      ;wersja bez skoku            ;jle  pomin
      cmovg rax, [rsi + 8*r11 - 8] ;mov rax, [rsi + 8*r11 - 8]
                                   ;pomin:
      dec r11
      jnz petlaK
      dec r10
      jnz petlaN
      dec rdx
      jnz petlaM
      ret
minM3 endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj. Określ zawartości wszystkich rejestrów.

Napisz funkcję poszukującą maksimum macierzy trójwymiarowej elementów typu int.

5. Iloczyn macierzy M=U*V

Przygotuj 3 macierze w x k, w x n i n x k liczb typu INT64. Zadeklaruj funkcję iloczynu:

```
extern "C" void iloczynMeUxV(INT64**, INT64**, INT64**, INT64, INT64, INT64);
```

Wpisz funkcję sumy w pliku .asm:

```
.code

iloczynMeUxV proc uses rbx rsi rdi, mm:ptr, uu:ptr, vv:ptr, w:qword, k:qword, n:qword
;RCX = **mm, RDX = **uu, R8 = **vv, R9 = w, Stos = k, n;
petlaw:
      mov   rsi, [rdx + 8*r9 - 8]
      mov   rdi, [rcx + 8*r9 - 8]
      mov   r10, k
petlak:
      mov   rax, 0
      mov   r11, n
petlan:
      mov   rbx, [r8 + 8*r11 - 8]   ;nty wiersz
      mov   rbx, [rbx + 8*r10 - 8]  ;kta kolumna
      imul  rbx, [rsi + 8*r11 - 8]  ;x nty element wtego wiersza
      add   rax, rbx
      dec   r11
      jnz   petlan

      mov   [rdi + 8 * r10 -8], rax
      dec   r10   ;k
      jnz   petlak

      dec   r9    ;w
      jnz   petlaw
      ret
iloczynMeUxV endp
```

Uruchom funkcję w trybie krokowym i przeanalizuj. **Zwróć uwagę na deklarację funkcji.**

Napisz samodzielnie mnożenie macierzy dla typu int (32 bity).

Zadania do samodzielnego wykonania na platformie x86 i x64

1. Wyznacz macierz transponowaną MxN dla macierzy NxM dla liczb typu INT oraz INT64.
2. Wyznacz sumę dwóch macierzy pod warunkiem, że wartość elementu w macierzy pierwszej jest większa niż odpowiadającego mu elementu w macierzy drugiej.
3. Wyznacz sumę dwóch macierzy pod warunkiem, że wartość elementu w macierzy pierwszej jest większa niż elementu w macierzy drugiej bez użycia instrukcji skoku dla warunku sprawdzania wartości elementu.
4. Podnieś do kwadratu wszystkie elementy macierzy znajdującej się na i ponad główną przekątną macierzy.
5. Wyzeruj co drugi element macierzy – wygląd finalny ma przypominać szachownicę. Zwróć szczególną uwagę na wartość indeksów (indeksy parzyste i nieparzyste).
6. Dla każdego wiersza w danej macierzy wyzeruj elementy o indeksach równych 2^k (np.: 0,1,2,4,8,16,32… dla $k < 64$).
7. Spróbuj wykonać transponowanie macierzy dla macierzy typu Float i Double – czy jest to możliwe na procesorze?