# Лабораторная работа 4 (0100 = 4)

# Использование ассемблерных вставок в программах на C++. Команды пересылки

## Задание на лабораторную работу

### Задание Л4.з1.

Как в задании Л1.з4, создайте массивы $Ms$ из 16-битных целых чисел, $Ml$ из 32-битных целых чисел, $Mq$ из 64 - битных целых чисел (длина и начальные значения аналогичны Л1.з4).

Реализуйте для каждого массива $M$ вставку, записывающую непосредственное значение 16 в $M$ [$i$] для заданного $i \in [0, N)$ с использованием команды $mov$, где выражение $M$ [$i$] является выходным параметром вставки в памяти. Так как оба операнда $mov$ здесь не имеют определённого размера (непосредственное значение и память), необходимо указывать для $mov$ суффикс размера: $movw$, $movl$, $movq$.

Здесь и далее все целочисленные массивы до и после изменения выводите в шестнадцатеричном представлении.

```
#include <iostream>
#define N 5

#define PRINT_i(arr,type)\
    std::cout << "\n"#arr " in hex:     "; \
    for (int i = 0; i < N; i++) { printf("%#0*llx ", (int)(sizeof(type)*2 + 2), static_cast<lon
    std::cout << std::endl;

int main(){
    short xs=0xEADE;
    int xl = 0xADE1A1D;
    long long xq = 0xC1A551F1AB1E;
    short Ms[N]{xs,xs,xs,xs,xs};
    int Ml[N]{xl,xl,xl,xl,xl};
    long long Mq[N]{xq,xq,xq,xq,xq};

    std::cout<<"\n\nshort"<<std::endl;
    PRINT_i(Ms,short);
    asm volatile(
        "movw $16, %0;\n"
        : "=m" (Ms[2])
    );
    PRINT_i(Ms,short);

    std::cout<<"\n\nint"<<std::endl;
    PRINT_i(Ml,int);
    asm volatile(
        "movl $16, %0;\n"
        : "=m" (Ml[2])
    );
    PRINT_i(Ml,int);

    std::cout<<"\n\nlong long"<<std::endl;
    PRINT_i(Mq,long long);
    asm volatile(
        "movq $16, %0;\n"
        : "=m" (Mq[2])
    );
    PRINT_i(Mq,long long);
```

```
    return 0;
}
```

## Задание Л4.з2.

Реализуйте для одного из массивов $M$ (по варианту согласно таблице Л4.1) вставку, записывающую непосредственное (−1) в $M$ [$i$], где адрес начала массива $M$ и индекс $i$ передаются как входные параметры в регистрах.

Используйте компоненты эффективного адреса ($Base$, $Index$, $2^{Scale}$). Разрядность компонент $Base$ и $Index$ должна быть одинаковой, поэтому для переносимости вставки необходимо объявить переменную $i$ не как $int$ (4 байта как для 32-, так и для 64-битного режимов), а как $size\_t$ (размер равен размеру указателя).

**Варианты целочисленного массива $M$**

Таблица Л4.1

| $(№ - 1)\%3 + 1$ | **Вариант** |
|---|---|
| 1 | $Ms$ |
| 2 | $Ml$ |
| 3 | $Mq$ |

```
#include <iostream>

#define N 5

#define PRINT_i(arr, type) \
    std::cout << "\n" #arr " in hex:     "; \
    for (int i = 0; i < N; i++) { \
        printf("%#0*llx ", (int)(sizeof(type) * 2 + 2), static_cast<long long>(arr[i])); \
    } \
    std::cout << std::endl;

int main() {
    int xl = 0xADE1A1D;
    int Ml[N]{xl, xl, xl, xl, xl};
    size_t index = 2;
    int *base = Ml;

    std::cout << "\n\nint" << std::endl;
    PRINT_i(Ml, int);

    asm volatile(
        "movl $-1, (%0,%1,4)\n\t"
        :
        : "r"(base), "r"(index)
        :
    );

    PRINT_i(Ml, int);

    return 0;
}
```

## Задание Л4.з3.

Реализуйте вставку, записывающую непосредственное значение 0xBB в заданный байт $M\ q[i]$ (по варианту согласно таблице Л4.2; младший байт считайте нулевым) с использованием одной команды $mov$ ($movb$) и всех компонент эффективного адреса $Disp(Base, Index, 2Scale)$; адрес начала массива $M_q$ и индекс $i$ передаются как входные параметры в регистрах.

**Варианты перезаписываемого байта $Mq[i]$ для Л4.з3**

Таблица Л4.2

| $(№ - 1)\%5 + 1$ | **Вариант** |
|---|---|
| 1 | Первый байт после младшего |
| 2 | Третий байт |
| 3 | Четвёртый байт |
| 4 | Шестой байт |
| 5 | Седьмой байт (старший байт 64-битного $Mq[i]$) |

```
#include <iostream>

#define N 5

#define PRINT_i(arr, type) \
    std::cout << "\n" #arr " in hex:     "; \
    for (int i = 0; i < N; i++) { \
        printf("%#0*llx ", (int)(sizeof(type) * 2 + 2), static_cast<long long>(arr[i])); \
    } \
    std::cout << std::endl;

int main() {
    long long xq = 0xC1A551F1AB1E;
    long long Mq[N]{xq,xq,xq,xq,xq};
    size_t index = 2;
    long long *base = Mq;

    std::cout << "\n\nint" << std::endl;
    PRINT_i(Mq, long long);

    asm volatile(
        "movb $0xBB, 0x3(%0,%1,8)\n\t"
        :
        : "r"(base), "r"(index)
        :
    );

    PRINT_i(Mq, long long);

    return 0;
}
```

## Задание Л4.з4.

Реализуйте вставку, записывающую в $M$ [$i$] значение $x$ ($M$ по варианту согласно таблице Л4.1; размер переменной $x$ равен размеру элемента $M$ ),где значение $x$ передаётся как входной параметр в памяти, $M$ и $i$ — как входные параметры в регистрах.

Так как команда x86 не может адресовать два операнда в памяти, прямая пересылка $x \rightarrow M$ [$i$] невозможна; используйте промежуточный регистр (таблица Л4.3).

**Варианты временного РОН**

Таблица Л4.3

| $(№ - 1)\%7 + 1$ | **Вариант** |
|---|---|
| 1 | Регистр $A$ ($al/ax/eax/rax$) |
| 2 | Регистр $C$ ($cl/cx/ecx/rcx$) |
| 3 | Регистр $D$ ($dl/dx/edx/rdx$) |
| 4 | Регистр $si$ ($sil/si/esi/rsi$) |
| 5 | Регистр $di$ ($dil/di/edi/rdi$) |
| 6 | Регистр $r8$ ($r8b/r8w/r8d/r8$) на 64-битной платформе, $A$ на 32 |
| 7 | Регистр $r9$ ($r9b/r9w/r9d/r9$) на 64-битной платформе, $C$ на 32 |

```
#include <iostream>

#define N 5

#define PRINT_i(arr, type) \
    std::cout << "\n" #arr " in hex:     "; \
    for (int i = 0; i < N; i++) { \
        printf("%#0*llx ", (int)(sizeof(type) * 2 + 2), static_cast<long long>(arr[i])); \
    } \
    std::cout << std::endl;

int main() {
    int xl = 0xADE1A1D;
    int Ml[N]{xl, xl, xl, xl, xl};
    int x=8;
    size_t index = 2;
    int *base = Ml;

    std::cout << "\n\nint" << std::endl;
    PRINT_i(Ml, int);

    asm volatile(
        "movl %2, %%ecx\n\t"
        "movl %%ecx, (%0,%1,4)\n\t"
        :
        : "r"(base), "r"(index),"m"(x)
        : "ecx", "memory"
    );

    PRINT_i(Ml, int);

    return 0;
}
```

## Задание Л4.з5.

Реализуйте вставку, записывающую в $M$ [$i$] значение $x$ аналогично Л4.з4, но во вставку передаётся адрес &$x$.

```
#include <iostream>

#define N 5

#define PRINT_i(arr, type) \
    std::cout << "\n" #arr " in hex:     "; \
    for (int i = 0; i < N; i++) { \
        printf("%#0*llx ", (int)(sizeof(type) * 2 + 2), static_cast<long long>(arr[i])); \
    } \
    std::cout << std::endl;

int main() {
    int xl = 0xADE1A1D;
    int Ml[N]{xl, xl, xl, xl, xl};
    int x = 8;
    size_t index = 2;
    int *base = Ml;

    std::cout << "\n\nint" << std::endl;
    PRINT_i(Ml, int);

    asm volatile(
        "movl (%2), %%ecx\n\t"
        "movl %%ecx, (%0,%1,4)\n\t"
        :
        : "r"(base), "r"(index), "r"(&x)
        : "ecx", "memory"
    );

    PRINT_i(Ml, int);

    return 0;
}
```

## Задание Л4.з6.

Реализуйте вставку, рассчитывающую для целочисленных $x$ и $y$ значения $z = x + y$ и $w = x - y$ при помощи команд $add$ и $sub$. Разрядность указана в таблице Л4.4; переменные $x$, $y$, $z$, $w$ передаются во вставку как параметры ($z$ и $w$ — выходные, $x$ и $y$ — входные).

**Варианты разрядности $x, y, z, w$**

Таблица Л4.4

| $(№ - 1)\%2 + 1$ | **Вариант** |
|---|---|
| 1 | 64 бита |
| 2 | 16 бит |

```
#include <iostream>


int main() {
    short x=8;
    short y=9;
    short z=0;
    short w=0;

    asm volatile(
        "movw %2,%%ax\n\t" // x to eax
        "movw %3, %%cx\n\t" // y to ecx
        "add %%ax,%0 \n\t" // z = z + x
        "add %%cx,%0 \n\t" // z = z + y
        "add %%ax,%1 \n\t" // w = w + x
        "sub %%cx,%1 \n\t" // w = w + y
        : "=m"(z),"=m"(w)
        : "m"(x), "m"(y)
        : "ax", "cx", "memory"
    );
    std::cout<<"z = "<<z<<std::endl;
    std::cout<<"w = "<<w<<std::endl;
    return 0;
}
```

## Задание Л4.з7.

Определите, доступны ли на выбранной платформе расширения AVX и SSE, используя команду *cpuid* или документацию на процессор.

Как в задании Л1.з4, создайте массивы $Mfl$ из 64- битных чисел с плавающей запятой и $Mfs$ из 32-битных чисел с плавающей запятой.

Реализуйте вставку, записывающую в $M$ [$i$] значение $x$ с плавающей запятой аналогично Л4.з4 ($M$ по варианту согласно таблице Л4.5; размер переменной $x$ равен размеру элемента $M$ ; $x$, $M$ и $i$ — параметры вставки), используя команды AVX $vmovsd/vmovss$ или их SSE-аналоги $movsd/movss$. Используйте промежуточный регистр $xmm$ i, где номер регистра $i \in [0, 5]$ рассчитывается как $(№ - 1)\%6$ (по варианту).

**Варианты массива $M$ из значений с плавающей запятой**

Таблица Л4.5

| $(№ - 1)\%2 + 1$ | **Вариант** |
|---|---|
| 1 | $Mfl$ |
| 2 | $Mfs$ |

```
#include <iostream>
#define N 5
#define PRINT_f(arr,type)\
    std::cout << "\n\n"#arr " in decimal: "; \
    for (int i = 0; i < N; i++) { printf("%.2f ", arr[i]); } \
    std::cout<<std::endl;
int main() {
    bool AVX_bit;
    asm(
        "cpuid\n"
        "test $(1 << 28), %%ecx\n"
        : "=@ccnz"(AVX_bit)
        : "a"(1)
        : "ebx", "ecx", "edx", "cc"
    );
    std::cout<<"AVX_bit = "<<AVX_bit<<std::endl;
    std::cout<<"AVX is available"<<std::endl;

    float x1 = -8.0f/5.0f;
    float Mfs[N]{x1,x1,x1,x1,x1};
    PRINT_f(Mfs,float);
    float x=4.8;
    size_t index = 2;
    float *base = Mfs;
    asm(
        "movss %2, %%xmm1\n\t"
        "movss %%xmm1, (%0,%1,4)\n\t"
        :
        : "r"(base), "r"(index),"m"(x)
        : "xmm1", "memory"
    );
    PRINT_f(Mfs,float);

    return 0;
}
```

## Задание Л4.з8.

Реализуйте вставку, записывающую в $M\ [i]$ значение с плавающей запятой, равное целочисленному значению $x$. Преобразование целочисленного $x$ к нужному виду выполните при помощи команд AVX $vcvtsi2sd\ /\ vcvtsi2ss$ или их SSE-аналогов $cvtsi2sd\ /\ cvtsi2ss$.

```cpp
#include <iostream>
#define N 5
#define PRINT_f(arr,type)\
   std::cout << "\n\n"#arr " in decimal: "; \
   for (int i = 0; i < N; i++) { printf("%.2f ", arr[i]); } \
   std::cout<<std::endl;

int main() {



   float x1 = -8.0f/5.0f;
   float Mfs[N]{x1,x1,x1,x1,x1};
   PRINT_f(Mfs,float);
   int x=4;
   size_t index = 2;
   float *base = Mfs;
   asm(
       "cvtsi2ss %2, %%xmm1\n\t"
       "movss %%xmm1, (%0,%1,4)\n\t"
       :
       : "r"(base), "r"(index),"m"(x)
       : "xmm1", "memory"
   );
   PRINT_f(Mfs,float);

   return 0;
}
```