Отчет по 5 лабораторной работе.

Команда: Холодов А.В. Иванов А.А. ПИН-21

Задание Л5.з1. Реализуйте расчёт беззнакового целочисленного выражения (таблица Л5.1) как ассемблерную вставку в программу на C/C++.

2	$\begin{cases} z = (x+34)/y \\ w = (x+34)\%y \end{cases}$

Реализация:

```
void task1(){
    unsigned int x,y,z,w;
    x = 35;
    y = 2;
    asm(
    "movl %[X], %%eax\n\t"
    "movl %[Y], %%ebx\n\t"
    "addl $34, %%eax\n\t"
    "XOR %%edx, %%edx\n\t" //обнуление регистра
    "divl %%ebx\t"
    :"=a"(z), "=d"(w)
    : [X] "m"(x), [Y] "m"(y)
    :"cc", "ebx");
    cout << hex << z << ' ' << hex << w << endl;
    // проверка
    z = (x+34)/y;
    w = (x+34)%y;
    cout << hex << z << ' ' << hex << w << endl;
}
```

Вывод:

```
Task 1:
22 1
22 1
```

Задание Л5.з2. Реализуйте задание Л5.з1, располагая параметры [X] и [Y] *в регистрах общего назначения* ([X] в регистре A, [Y] — в выбираемом компилятором).

Как выглядит обращение к параметрам [X] и [Y]? В какой регистр попал [Y]?

Реализация:

```
void task2(){
    unsigned int x,y,z,w;
    x = 35;
    y = 2;
    asm(
    "XOR %%edx, %%edx\n\t"
    "addl $34, %%eax\n\t"
    "divl %[Y]\t"
    :"=a"(z), "=&d"(w)
    : [X] "a"(x), [Y] "r"(y)
    :"cc"
    );
    cout << z << ' ' << w << endl;
    // проверка
    z = (x+34)/y;
    w = (x+34)%y;
    cout << z << ' ' << w << endl;
}
```

Вывод:

```
Task 2:
22 1
22 1
```

Задание Л5.з3. Реализуйте задание Л5.з1, передав вставке не значения x и y, а указатели $p = \&x$ и $q = \&y$ (параметры [pX] и [pY]), разместив [pX] и [pY] в регистрах общего назначения, выбираемых компилятором.

Реализация:

```
void task3(){
    unsigned int x,y,z,w;
    x = 35;
    y = 2;
    unsigned int* p = &x;
    unsigned int* q = &y;
    asm (
    "XOR %%edx, %%edx\n\t"
    "movl (%[pX]), %%eax\n\t"
    "addl $34, %%eax\n\t"
    "divl (%[pY])"
    :"=&a"(z), "=&d"(w)
    :[pX] "r"(p), [pY]"r"(q)
    :"cc"
    );
    cout << z << ' ' << w << endl;
    // проверка
    z = (x+34)/y;
    w = (x+34)%y;
    cout << z << ' ' << w << endl;
}
```

Вывод:

```
Task 3:
22 1
22 1
```

Задание Л5.з4. Реализуйте задание Л5.з1 для 16-битных x, y, z, w (не расширяя их до 32 бит).

Реализация:

```
void task4(){
    unsigned short x,y,z,w;
    x = 35;
    y = 2;
    asm (
    "XOR %%dx, %%dx\n\t"
    "mov %[X], %%ax\n\t"
    "mov %[Y], %%bx\n\t"
    "add $34, %%ax\n\t"
    "div %%bx\t"
    :"=a"(z), "=d"(w)
    : [X] "m"(x), [Y] "m"(y)
    :"cc", "ebx");
    cout << z << ' ' << w << endl;
    // проверка
    z = (x+34)/y;
    w = (x+34)%y;
    cout << z << ' ' << w << endl;
}
```

Вывод:

```
Task 4:
22 1
22 1
```

Задание Л5.з5. На языке C/C++ выделите память под массив M (статический или динамический) из N 32-битных целых чисел и инициализируйте M значениями 0x44332211.

Реализуйте для заданного $k \in [0, N)$ запись значения $x \neq 0$ на место элемента $M[k]$, используя компоненты эффективного адреса $(Base, Index, 2^{Scale})$.

Примечание: разрядность компонент $Base$ и $Index$ должна быть одинаковой, поэтому для переносимости вставки необходимо объявить переменную k не как int (4 байта как для 32-, так и для 64-битного режимов), а как $size\_t$ (размер равен размеру указателя).

Массив до и после записи выводите в шестнадцатеричном представлении.

Реализация:

```
void task5(){
    int const N = 5;
    int M[N];
    for (int i = 0; i < N; i++)
        M[i] = 0x44332211;
    cout << "Data: ";
    for (int i = 0; i < N; i++)
        printf("%#10X ", M[i]);
    cout << endl;
    size_t k = 2; // позиция
    int x = 7; // число
    asm (
    "movl %0, (%1, %2, 4)\n"
    :
    :"r"(x), "r"(M) ,"r"(k)
    : "memory"
    );
    cout << "Data: ";
    for (int i = 0; i < N; i++)
        printf("%#10X ", M[i]);
}
```

Вывод:

```
Task 5:
Data: 0X44332211 0X44332211 0X44332211 0X44332211 0X44332211
Data: 0X44332211 0X44332211        0X7 0X44332211 0X44332211
```

Задание Л5.з6. Реализуйте для заданного $j \in [0, N)$, $j \neq k$ запись значения FF в старший байт элемента $M[j]$, используя все компоненты эффективного адреса.

Реализация:

```
void task6(){
    int const N = 5;
    unsigned int M[N];
    for (int i = 0; i < N; i++)
        M[i] = 0x44332211;
    size_t j = 4;
    cout << "Data: ";
    for (int i = 0; i < N; i++)
        printf("%#10X ", M[i]);
    cout << endl;
    asm(
    "movb $0xFF, 3(%0, %1, 4)\n"
    :
    : "r"(M) ,"r"(j)
    : "memory"
    );
    cout << "Data: ";
    for (int i = 0; i < N; i++)
        printf("%#10X ", M[i]);
}
```

Вывод:

```
Task 6:
Data: 0X44332211 0X44332211 0X44332211 0X44332211 0X44332211
Data: 0X44332211 0X44332211 0X44332211 0X44332211 0XFF332211
```

Л5.3. Вопросы

1. Каким ключевым словом открывается ассемблерная вставка?
2. Где описываются выходные параметры ассемблерных вставок расширенного синтаксиса GCC? Что означают символы =, =&, + в начале строки ограничений выходного параметра?
3. Где описываются входные параметры?
4. Где указывается список перезаписываемых [clobbers] во вставке регистров (кроме параметров)? Какая строка соответствуют изменению флагов $flags$? Какая строка соответствуют изменению памяти (кроме параметров)?
5. Где описываются метки ЯВУ, на которые может быть передано управление из вставки?
6. Какое ключевое слово нужно указать после `asm`, чтобы запретить компилятору оптимизировать вставку?
7. Какое ключевое слово нужно указать после `asm`, чтобы показать, что управление из вставки может передаваться на ту или иную метку C/C++ (зато у этой вставки нет выходных параметров)?

1.

Синтаксис оператора ассемблерной вставки следующий.

`__asm__` (***вставка*** : ***список_выходных_операндов*** : ***список_входных_операндов*** : ***список_разрушаемых_регистров***);

Начинается оператор ассемблерной вставки с ключевого слова `asm` или `__asm__`, после чего в круглых скобках следует ее описание.

2.

=/=&/+: инициализация и совмещение регистровых выходных параметров. Если при этом ограничение выходного параметра `[X]` в регистре ρ начинается с:

=, то регистр ρ никак не инициализируется перед вставкой, а *в том же регистре ρ может быть размещён какой-либо из входных параметров* (скорее всего в регистре ρ *будет* размещён один из входных параметров, так как компилятор стремится минимизировать общее число задействованных во вставке регистров);

=&, то регистр ρ никак не инициализируется перед вставкой (аналогично =), но (в отличие от =) в регистре ρ не может быть размещён никакой другой параметр (см. раздел 4.3.6);

+, то регистр ρ инициализируется перед вставкой значением `cvariablename` (значение `cvariablename` копируется в ρ), и в регистре ρ не может быть размещён никакой другой параметр.

3. Раскрыт в 1 вопросе.

4.

Перезаписываемые элементы (clobbers)

Код в ассемблерной вставке может прямо или косвенно изменять значения не только параметров, но и прочих регистров, других участков памяти, а также значения регистра флагов.

Для указания компилятору, какие именно элементы (кроме параметров) меняет вставка, используется список перезаписываемых (clobber, «сбитых») элементов [24].

Если во вставке явно или неявно модифицируется какой-либо регистр, его имя в виде строки необходимо указать в списке перезаписываемых элементов. Имя может быть указано как с префиксом % (в частности, `"%eax"`), так и без него (как `"eax"`). Если этого не сделать, компилятор, размещая параметры с ограничениями `"r"`, `"g"` и т. п., может поместить их в эти регистры, и значение такого параметра будет изменяться непредсказуемо.

Если во вставке изменяется значение в памяти, не считая явно указанных выходных параметров (например, записываются данные в массив), в список перезаписываемых элементов необходимо добавить специальное значение `"memory"`.

При указании `"memory"` в списке перезаписываемых элементов вставка будет барьером памяти: все операции работы с памятью, которые были в программе до ассемблерной вставки, выполнятся до неё, а те, что стоят в программе после — будут после. В противном случае компилятор может менять местами как команды,

реализующие операторы C++, так и команды ассемблерной вставки (в том числе перемешивая их).

Если во вставке изменяется значение регистра флагов (в частности, флаги меняют все арифметические инструкции), в список перезаписываемых элементов необходимо добавить специальное значение `"cc"`.

5.

На практике описанную выше базовую форму ассемблерных вставок (без параметров) не стоит использовать *никогда*.
Только расширенная форма даёт возможность корректного взаимодействия с программой на ЯВУ.

Расширенная форма с выходными параметрами

Расширенных форм вставки в GCC две. Первая (с выходными и входными параметрами) выглядит следующим образом (листинг 4.15).

Листинг 4.15. Расширенная форма с выходными параметрами

```
1 asm [volatile] (
2     "команды␣и␣директивы␣ассемблера"
3     "как␣последовательная␣текстовая␣строка"
4     : [<выходные параметры>] : [<входные параметры>] :
      [<перезаписываемые элементы>]
5     );
```

Ключевое слово `volatile` используется для того, чтобы указать компилятору, что вставляемый ассемблерный код может обладать побочными эффектами, поэтому попытки оптимизации могут привести к логическим ошибкам.

6.

Параметры (как входные `[X]` и `[A]`, так и выходной `[Y]`) расположены в памяти. К константе a (значению входного параметра `[A]`) применён модификатор `volatile`, чтобы компилятор не оптимизировал её и разместил в памяти, как и необходимо.

7.

Для этого понадобится вторая форма расширенной ассемблерной вставки, с входными параметрами и метками выхода — *asm goto*. У такой вставки нет выходных параметров (отсутствует секция между первыми двумя двоеточиями), но присутствует дополнительная секция меток (четвёртое двоеточие, после которого следует перечень меток, на которые можно передать управление из вставки).

Внутри вставки к меткам можно обратиться по номеру параметра-метки (сквозному с нуля): $\%\ell 1$ для `has_avx`, $\%\ell 2$ для `has_sse4` (номер 0 соответствует входному параметру в A). Если управление не будет передано ни на одну из меток (в данном случае — если не поддерживается ни AVX, ни SSE4.3), после вставки будет выполнена следующая за ней команда (строка 13 листинга).