

Ассемблерные вставки в компиляторе gcc

Ассемблерные вставки используются для «насилованного» размещения в Си-программах ассемблерного кода, явно заданного программистом. Содержимое ассемблерной вставки никак компилятором не анализируется, но имеется возможность описать то, как это содержимое взаимодействует с переменными Си-программы и как изменятся регистры после выполнения этого ассемблерного кода.

Замечание. Важно помнить, что после компиляции Си-программы, код ассемблерной вставки будет транслироваться `gas`, а не `nasm`. Естественным синтаксисом `gas` является AT&T, существенно отличающийся от синтаксиса Intel. Современные версии `gas` поддерживают синтаксис Intel, переключиться на него можно специальной директивой, но диалект поддерживаемого синтаксиса будет отличаться от диалекта `nasm` рядом особенностей.

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.

вставка представляет собой строковую константу с ассемблерными инструкциями. В теле вставки могут находиться не только ассемблерные инструкции, но и вообще любые директивы, распознаваемые ассемблером `gas`. В частности, это позволяет изменить используемый им синтаксис инструкций по-умолчанию.

Пример 1

```
__asm__ ( ".intel_syntax noprefix\n\t"
"mov eax, %1\n\t"
"mov %0, eax\n\t" /* ассемблерная вставка */
: "=r"(b) /* выходные операнды */
: "r"(a) /* входные операнды */
: "%eax" /* разрушаемые регистры */
);
```

Директива `.intel_syntax` меняет синтаксис AT&T на синтаксис Intel; необходимо дополнительно указывать смену синтаксиса для операндов инструкций, `noprefix`, что позволит писать код в более близком к диалекту `nasm` виде, не используя при записи имен регистров префикс `%`.

Допустима сокращенная версия вставки, состоящая только из строки с командами:

```
asm("hlt\n\t");
```

Для связи ассемблерных инструкций с переменными Си-программы используются следующие за **вставкой** два элемента оператора: списки операндов, в которых операнды перечислены через запятую. Каждый описанный операнд затем может использоваться в ассемблерных инструкциях, обращение к нему осуществляется по номеру с префиксом `%`. Нумерация начинается с 0, и идет непрерывно, объединяя все элементы списков выходных и входных операндов.

Операнд имеет следующий вид:

ограничение_типа (*имя_переменной*)

имя_переменной – не что иное, как имя Си-переменной, значение которой вы хотите использовать в ассемблерном коде.

ограничение_типа – строковая константа, описывает допустимый тип операнда.

В приведенном выше примере ассемблерные инструкции используют операнды 0 и 1, приведенные в двух списках.

```
"mov eax, %1\n\t"
"mov %0, eax\n\t"
```

В первой инструкции в регистр `eax`, помещается значение операнда вставки №1, который является входным. Во второй инструкции в выходной операнд вставки (№0, т.к. первым идет список выходных операндов), пересылается значение регистра `eax`.

Для выходных операндов строка ограничения типа должна начинаться с символа ``='`. Следующий в строке символ указывает (кодирует), куда компилятору нужно поместить значение соответствующей переменной. Способ кодировки одинаковый для входных и выходных операндов вставки.

```
: "=r"(b)          /* выходные операнды */
: "r"(a)           /* входные операнды */
```

Для рассматриваемого примера описание операндов требует размещения значения Си-переменной `a` на каком-либо регистре общего назначения (задано символом `r`), перед тем как начнет выполняться код вставки. После того как код вставки завершится, значение переменной `b` будет находиться на регистре общего назначения и его потребуется записать в переменную `b`.

Пример того, как компилятор организует связь вставки и Си-кода можно увидеть ниже.

gcc -std=c99 -Wall -O0 -S -masm=intel asm_inline.c	
<pre>int f() { int a=10, b; __asm__ (".intel_syntax noprefix\n\t" "mov eax, %1\n\t" "mov %0, eax\n\t" ": "=r"(b) : "r"(a) : "%eax"); return b; }</pre>	<pre>f: push ebp mov ebp, esp sub esp, 16 mov DWORD PTR [ebp-4], 10 mov edx, DWORD PTR [ebp-4] #APP # 3 "asm_inline.c" 1 .intel_syntax noprefix mov eax, edx mov edx, eax # 0 "" 2 #NO_APP mov DWORD PTR [ebp-8], edx mov eax, DWORD PTR [ebp-8] leave ret</pre>

Замечание. Несмотря на ключ `-masm=intel`, синтаксис полученного кода несколько отличается от синтаксиса `nas`: после ключевого слова, описывающего размер операнда инструкции, `gcc`

помещает ключевое слово PTR. Другие различия диалектов синтаксиса Intel ассемблеров gas и nasm в данном фрагменте кода не проявляются.

Как видно в ассемблерном листинге, gcc перед выполнением ассемблерной вставки поместил значение переменной a на один из доступных регистров общего назначения, в данном случае – EDX. После того, как вставка закончила работать, этот же регистр использовался для пересылки значения выходного операнда в переменную b.

Ограничение типа позволяет в коде ассемблерной вставки использовать операнды согласно допустимым форматам команд. Если требуется, что бы операнд был размещен в памяти, вместо символа r следует использовать символ m.

В некоторых случаях требуется не только переслать значение переменной на регистр, но и обеспечить, чтобы это был конкретный регистр общего назначения. В этом случае вместо символа r используются следующие коды.

Символьный код	Регистры
a	eax, ax, al
b	ebx, bx, bl
c	ecx, cx, cl
d	edx, dx, dl
S	esi, si
D	edi, di

Размер регистра будет выбран компилятором автоматически, исходя из размера данных.

Следующий пример показывает, как может быть реализовано копирование массивов целых чисел ассемблерной вставкой.

Пример 2

```
__asm__ __volatile__ ( "cld\n\t"
                      "rep movsl\n\t"
                      :
                      : "S" (src), "D" (dest), "c" (numwords)
                      : "%ecx", "%esi", "%edi"
                      )
```

Замечание. Еще одно различие диалектов gas и nasm заключается в форме записи строковых инструкций: отличаются суффиксы, кодирующие размер обрабатываемых данных. Для ассемблера nasm корректна запись "rep movsd".

В рассматриваемом примере нет явных операндов ассемблерных инструкций, зато есть три операнда ассемблерной вставки: src, dest, numwords. Они описывают начальные адреса массивов и количество копируемых элементов. Выходных операндов у вставки нет: все изменения состояния программы происходят в памяти. Входные операнды, связанные с соответствующими переменными, будут помещены в регистр ESI, EDI и ECX.

Компилятору о побочном эффекте, возникающем из-за копирования, необходимо сообщить ключевым словом __volatile__ после ключевого слова __asm__. Это обезопасит корректность программы, поскольку в отсутствии этого ключевого слова некоторые оптимизирующие преобразования потенциально могут перемещать код, в том числе и код ассемблерных вставок.

Последний элемент оператора вставки, **список_разрушаемых_регистров**, сообщает компилятору о тех регистрах, которые будут «разрушены» в результате выполнения кода вставки. В данном случае это три уже перечисленных выше регистра ESI, EDI и ECX.

В первом рассмотренном примере регистры для связи операндов с переменными выбирались автоматически, в список разрушаемых регистров следует отнести только регистр eax.

Возможности оператора вставки сильно выходят за рамки рассмотренных в данном описании примеров. Полная техническая информация об особенностях работы оператора вставки доступна в документации компилятора gcc соответствующей версии.

Важно помнить, что ассемблерная вставка негативно сказывается на производительности программы. Связывание операндов вставки и переменных Си-программы требует дополнительных инструкций копирования, но, самое главное, вставки меняют области применения оптимизирующих преобразований кода, не позволяя им достичь должных результатов.