## *Соглашения о вызовах*

Соглашения о вызовах - протокол взаимодействия вызывающей и вызываемой программ.

Необходимо согласовать следующие правила:

1. Способ передачи параметров (через стек, через регистры, смешанный; а также используемые регистры и их порядок).

2. Порядок размещения элементов в стеке

Порядок Pascal - первый параметр помещается в стек первым.

Порядок C - первый параметр помещается последним непосредственно перед адресом возврата.

3. Как передаётся указатель this (для методов объекта).

4. Какие регистры могут изменяться подпрограммой

5. Кто очищает стек и сохраняет/восстанавливает регистры

6. Инструкции вызова и возврата из подпрограмм.

7. Возврат значения из подпрограммы (функции).

## *Передача и возвращение управления из подпрограммы*

*На платформе x86 для вызова и возврата из подпрограммы используются соответственно команды call и ret; а значение обычно возвращается через регистр A .*

*Параметры обычно передаются либо через стек, либо смешанным способом: первые из тех, что можно разместить в отведённых регистрах, передаются через регистры, оставшиеся — через стек.*
*Остальные пункты по-разному реализованы в различных языках, компиляторах, операционных системах и для различной разрядности.*

*Пусть следующая команда, расположенная по адресу ci— call f (рис. 4.4, а). Команда call помещает в стек адрес следующей по порядку команды (ci+1) — адрес возврата, после чего в указатель команд помещается адрес , так что эта команда становится следующей для исполнения процессором*

*Когда в процессе исполнения подпрограммы встретится команда ret, из стека извлекается верхнее машинное слово — там должен быть адрес возврата — и помещается в указатель команд ip. Соответственно, выполнение вызывающей программы продолжится со следующей за call команды.*

*Таблица*

*Если столбец «Параметры в регистрах» пуст, все параметры передаются через стек. Указатель this обычно передаётся первым параметром.*

## *Таблица 4.1 32bit*

| **Соглашение** | **Параметры в регистрах** | **Порядок** | **Очистка стека** |
|---|---|---|---|
| **cdecl** | | C | вызывающая программа |
| **pascal** | | Pascal | функция |
| **winapi (stdcall)** | | C | функция |
| **gnu** | | C | this — функция, остальные — вызывающая программа |
| **gnu fastcall** | *ecx*, *edx* | C | функция |
| **gnu regparm (3)** | *eax*, *edx*, *ecx* | C | функция |
| **Borland fastcall** | *ecx*, *edx* | Pascal | функция |
| **Microsoft fastcall** | *ecx*, *edx* | C | функция |

*Согласно Фогу, в тридцатидвухбитных программах, как в Microsoft Windows, так и в Unix-подобных операционных системах (GNU/Linux, BSD, Mac OS X), подпрограмма может изменять регистры eax,ecx,edx; регистры сопроцессора st(0) – st(7) и регистры расширений xmm/ymm /zmm . Непри косновенными должны остаться ebx,ebp,esi,edi.*

*64bit*

*На шестидесятичетырёхбитных платформах применяется всего два соглашения о вызовах (Они несовместимы между собой. Регистры для передачи параметров используются в указанном порядке. Указатель this передаётся первым параметром.*

Как видно из таблицы 4.2, в 64-битном режиме под разными операционными системами в подпрограмме необходимо сохранять и восстанавливать разные регистры.

Таблица 4.2

| **Соглашение** | **Параметры в регистрах** | **Порядок** | **Очистка стека** | **Изменяемые регистры** | **Неизменяемые регистры** |
|---|---|---|---|---|---|
| **Microsoft Windows,** компиляторы MinGW, Microsoft, Intel | $rcx/zmm0$, $rdx/zmm1$, $r8/zmm2$, $r9/zmm3$ | C | вызывающая программа | $rax, rcx, rdx$, $r8-r11$, $st(0)-st(7)$, $x/y/zmm$, кроме младших частей $6-15$ | $rbx, rbp$, $rsi, rdi$, $r12-r15$, $xmm6-xmm15$ |
| **GNU/Linux, BSD, Mac OS X,** компиляторы GCC, Intel | $rdi$, $rsi$, $rdx$, $rcx$, $r8$, $r9$, $zmm0-zmm7$ | C | вызывающая программа | $rax, rcx, rdx$, $rsi, rdi$, $r8-r11$, $st(0)-st(7)$, $x/y/zmm$ | $rbx, rbp$, $r12-r15$ |

## В GAS:

### Вызов подпрограммы в GAS

*На тридцатидвухбитной платформе в GCC используются соглашения о вызове gnu, cdecl, gnu fastcall, gnu regparm. Для внешних функций с отключённым декорированием (extern "C") применяется только cdecl, то есть:*

- *размещение аргументов исключительно в стеке, без использования регистров, причём аргументы, меньшие 4 байт, расширяются до 4 байт;*

- *размещение аргументов в стеке таким образом, что первый аргумент оказывается на вершине стека;*
- *очистка стека выполняется вызывающей программой, так что в функции аргументы должны не сниматься со стека, а копироваться оттуда.*

Размещение аргументов в стеке справа налево и очистка стека вызывающей программой позволяет определить функции с переменным количеством аргументов, такие, как printf и scanf из стандартной библиотеки C, но надо помнить о небезопасности таких функций.

---

## *Адрес возврата*

*При вызове функции в стек сначала помещаются аргументы в соответствии с соглашением о вызовах, а затем команда вызова кладёт сверху адрес возврата. Соответственно, когда функция получает управление, то первые четыре байта по адресу, хранящемуся в , будут содержать адрес возврата.*

*Далее идут аргументы функции. При использовании соглашения о вызовах cdecl непосредственно за адресом возврата (по адресу +4) будет находиться первый параметр, за ним идёт второй и т. д.*

*Регистры B,bp,si,di не должны изменяться подпрограммой. Возврат значения по возможности выполняется через регистры:*

- *eax если результат —указатель или целое число до 4 байт(если результат меньше 4 байт, старшую часть необходимо обнулить);*
- *пара регистров edx : eax , если результат — целое число размером 8 байт;*
- *вершина стека сопроцессора, если результат — вещественное число; если результат не помещается в регистры, возвращается указатель на него (через eax).*

## *Команды ассемблера x86*

| Команда | Действие |
|---|---|
| nop<br>nop srm | Ничего не делает (no operation) |
| mov src, dest | Присваивание = (move) |
| movabs imm64, dreg64 | В 64-битном режиме присваивание абсолютного 64-битного адреса 64 = 64 |
| lea smem, dreg | Вычисление адреса и запись его в = & (load efective address) |
| xchg srm, dest | Обмен значений и |
| Работа со стеком | |
| push src | Помещение в стек (уменьшает указатель стека) |
| pop dest | Выталкивание значение из стека в (увеличивает указатель стека) |
| Вызов и возврат из функций | |
| call proc | Вызов подпрограммы — помещает в стек адрес следующей инструкции (адрес возврата) и переходит по адресу |
| ret [imm] | Возврат из подпрограммы — снимает со стека адрес возврата и помещает его в указатель команд.<br>Если указан параметр , снимает со стека ещё байтов. |

**Команды целочисленной арифметики**

Таблица 5.4

| Команда | Действие |
|---|---|
| **Сложение и вычитание** | |
| `inc dest` | Инкремент $++dest$ ($dest = dest + 1$, выполняется быстрее `add`) |
| `dec dest` | Декремент $--dest$ ($dest = dest - 1$, выполняется быстрее `sub`) |
| `add src, dest` | Сложение $dest \mathrel{+}= src$ $(dest = dest + src)$ |
| `adc src, dest` | Сложение с переносом из предыдущей части<br>$dest \mathrel{+}= src + CF$ $\left(dest = dest + (src + CF)\right)$ |
| `sub src, dest` | Вычитание $dest \mathrel{-}= src$ $(dest = dest - src)$ |
| `cmp src, dest` | Вычитание $dest - src$ без изменения $dest$ (только флаги) |
| `sbb src, dest` | Вычитание с переносом из предыдущей части<br>$dest \mathrel{-}= src + CF$ $\left(dest = dest - (src + CF)\right)$ |
| `neg dest` | Изменение знака $dest = -dest$ |
| **Расчёт линейной комбинации** | |
| `lea` $\delta$`(r1,r2,`$\gamma$`), dreg` | Вычисление эффективного адреса часто используется<br>для расчёта линейной комбинации $dreg = r1 + \gamma \cdot r2 + \delta$<br>$dreg, r1, r2$ — регистры, $\gamma$ — 1, 2, 4 или 8, $\delta$ — константа<br>`lea` не изменяет флагов |
| **Умножение и деление** | |
| `mul srm` | Беззнаковое умножение $D{:}A = A \cdot srm$ (таблица 5.5) |
| `imul srm` | Знаковое умножение $D{:}A = A \cdot srm$ (таблица 5.5) |
| `imul srm, dreg` | Умножение $dreg \mathrel{*}= srm$ $(dreg = dreg \cdot srm)$ |
| `imul imm, srm, dreg` | Знаковое умножение $dreg = imm \cdot srm$ |
| `div srm` | Беззнаковое деление с остатком (таблица 5.5)<br>$\begin{cases} A = (D{:}A)/srm \\ D = (D{:}A)\%srm \end{cases}$ |
| `idiv srm` | Знаковое деление с остатком (таблица 5.5)<br>$\begin{cases} A = (D{:}A)/srm \\ D = (D{:}A)\%srm \end{cases}$ |
| **Масштабирование (битовый сдвиг)** | |
| `shr times, dest` | Беззнаковое деление $dest \mathrel{/}= 2^{times}$ (остаток не вычисляется) |
| `sar times, dest` | Знаковое деление $dest \mathrel{/}= 2^{times}$ (остаток не вычисляется) |
| `shl times, dest`<br>`sal times, dest` | Умножение $dest \mathrel{*}= 2^{times}$ |

| Команда | Действие |
|---|---|
| movz srm, dreg | = с беззнаковым расширением (размер меньше ) |

| | |
|---|---|
| movs srm, dreg | = со знаковым расширением (размер меньше ) |
| cStD | Знаковое расширение регистра (таблица 5.7) |

---

## *Регистры общего назначения*

У процессора 8086 были A,B,C,D. От них идут все другие обозначения, т.е. ax,bx…
al - младшие 8 бит от ax
ah - старшие от ax

eax - 32bit
rax - 64bit

---

## *Флаги*

Все арифметические команды устанавливают по результатам вычислений флаги состояния.

Команды группы сложения/вычитания (add/sub, inc/dec и т.д) выставляют все шесть флагов состояния CF, PF, AF, ZF, SF, OF в соответствии с результатом:

ZF - флаг нуля , если результат равен нулю;

CF - флаг переноса (беззнакового переполнения) в случае переноса/заёма за пределы разрядной сетки (беззнакового переполнения);

SF - флаг знака, если старший (знаковый) бит результата равен 1;

OF - флаг знакового переполнения, если произошло знаковое переполнение (перенос/заём из знакового бита, но не за пределы разрядной сетки, или наоборот);

PF - флаг чётности, если количество единиц в младшем байте результата чётно;

AF - флаг вспомогательного переноса , если в младшем байте был перенос между тетрадами.  .

При этом add $-1, dest и sub $1, dest устанавливают флаги по-разному, в частности, при сложении числа -1 (что на 32-разрядной платформе равно 0xFFFFFFFF) с нулём не происходит переноса в старший бит (OF = 0); при вычитании единицы из нуля возникает заём из старшего бита ( OF = 1).

Побитовые команды (and, or, xor) выставляют флаги SF, ZF и PF  в соответствии с результатом аналогично группе сложения/вычитания, флаги переноса и знакового переполнения сбрасываются: CF= OF = 0. Значение флага AF не определено.

Команды умножения выставляют флаги CF = OF в зависимости от того, выходит ли результат за разрядность множителей. Значение остальных флагов не определено. После команд деления все шесть флагов имеют неопределённое значение.

*Вещественные числа можно сравнить командой fcom и подобными ей. После сравнения флаги состояния сопроцессора копируются в (вручную или автоматически—в зависимости от используемой команды сравнения) таким образом, что результат сравнения можно анализировать так же, как для целых беззнаковых чисел: ZF указывает на равенство, CF—на < ; кроме того, в PF копируется флаг несравнимости операндов.*

Кроме того, флаги можно установить или сбросить вручную с помощью специальных команд или загрузив изменённый регистр

## Команды обработки флагов

Таблица 5.9

| Команда | Действие |
|---|---|
| Установка отдельных флагов | |
| stc/clc/cmc | Установка (set, = 1)/сброс (clear, = 0)/инверсия ( =! ) флага переноса |
| std/cld | Установка/сброс флага направления |
| sti/cli | Установка/сброс флага разрешения прерываний |
| Обработка в целом или фрагментов | |
| lahf/sahf | Сохранение младшего байта в h/загрузка h в |
| pushf/pushfd/ pushfq | Загрузить младшие 16/32/64 бита в стек |
| popf/popfd/popfq | Выгрузить 16/32/64 бита из стека в (в младшую часть) |