

Компилятор C/C++ Clang
для DSP ELcore-30M
Соглашение о вызовах

ПОРЯДОК ИСПОЛЬЗОВАНИЯ ДОКУМЕНТА

Настоящая документация охраняется действующим законодательством Российской Федерации об авторском праве и смежных правах, в частности, законом Российской Федерации «Об авторском праве и смежных правах». ОАО НПЦ «ЭЛВИС» является единственным правообладателем исключительных авторских прав на настоящую документацию.

Настоящую документацию, не иначе как по предварительному согласию ОАО НПЦ «ЭЛВИС», запрещается:

- воспроизводить, т.е. изготавливать один или более экземпляров настоящей документации, ее части, в любой форме, любым способом;
- сдавать в прокат;
- публично показывать, исполнять или сообщать для всеобщего сведения,
- переводить;
- переделывать или другим образом перерабатывать (дорабатывать).

ОАО НПЦ «ЭЛВИС» оставляет за собой право в любой момент вносить изменения (дополнения) в настоящую документацию без предварительного уведомления о таком изменении (дополнении).

ОАО НПЦ «ЭЛВИС» не несет ответственности за вред, причиненный при использовании настоящей документации.

Передача настоящей документации не означает передачи каких-либо авторских прав ОАО НПЦ «ЭЛВИС» на нее.

Возникновение каких-либо прав на материальный носитель, на котором передается настоящая документация, не влечет передачи каких-либо авторских прав на данную документацию.

Все указанные в настоящей документации товарные знаки принадлежат их владельцам.

ОАО НПЦ «ЭЛВИС» ©, 2016

ОГЛАВЛЕНИЕ

ПОРЯДОК ИСПОЛЬЗОВАНИЯ ДОКУМЕНТА.....	2
Оглавление.....	3
1. ВСТУПЛЕНИЕ.....	4
2. ПРЕДСТАВЛЕНИЕ ДАННЫХ.....	5
3. ВЫРАВНИВАНИЕ СТЕКА	7
4. ИСПОЛЬЗОВАНИЕ РЕГИСТРОВ	8
5. ПОРЯДОК ВЫЗОВА ФУНКЦИИ	9
5.1. Функции с определенным числом параметров	10
5.2. Передача возвращаемого значения	10
5.3. Управление стеком	10
5.4. Вызов подпрограммы	11
5.5. Выход из подпрограммы.....	11
5.6. Указатель фрейма	11
5.7. Слот для указателя фрейма.....	12
5.8. Организация стека	12
5.9. Callee-saved registers (CSR).....	12
5.10. Функции с переменным числом параметров	12
6. СТРУКТУРА ФРЕЙМА. СТРУКТУРА ПРОЛОГА И ЭПИЛОГА.....	14
6.1. Функция с фиксированным числом аргументов без слотов для FP.....	14
6.2. Функция с фиксированным числом аргументов без слотов для FP.....	14
6.3. Функция с переменным числом параметров.....	15
Обозначения.....	17
ПРИМЕР 1. ELcore-30M. Уровень оптимизации –O0.	18
ПРИМЕР 2. ELcore-30M. Организация стека.	21
ПРИМЕР 3. Использование callee-saved registers.....	23
ПРИМЕР 4. Функция с переменным числом параметров.	25
ИСТОРИЯ ИЗМЕНЕНИЙ	29

1. ВСТУПЛЕНИЕ

Соглашение о вызовах - это часть двоичного интерфейса приложения, которая регламентирует особенности вызова подпрограммы, передачи аргументов и передачи результата выполнения подпрограммы.

2. ПРЕДСТАВЛЕНИЕ ДАННЫХ

Размер и выравнивание типов данных определяется согласно таблице 1.

Таблица 2.1. Размер и выравнивание стандартных типов данных DSP

Тип данных DSP	Размер в байтах	Размер в битах	Выравнивание в байтах
char, unsigned char	1	8	1
short, unsigned short	2	16	2
int, unsigned int	4	32	4
long, unsigned long	4	32	4
long long, unsigned long long	8	64	8
T*	4	32	4
float	4	32	4
double	4	32	4
long double	4	32	4

Тип `_Bool` расширяется до типа `char`.

Тип `double` (действительные числа с двойной точностью) приводится к типу `float` средствами компилятора.

Векторные типы данных выравниваются соответственно размеру. Векторные типы данных объявляются через расширения языка C.

```
typedef __attribute__((__vector_size__(2 * sizeof(short)))) short _v2i16;
typedef __attribute__((__vector_size__(4 * sizeof(short)))) short _v4i16;
typedef __attribute__((__vector_size__(8 * sizeof(short)))) short _v8i16;
typedef __attribute__((__vector_size__(2 * sizeof(int)))) int _v2i32;
typedef __attribute__((__vector_size__(4 * sizeof(int)))) int _v4i32;
typedef __attribute__((__vector_size__(2 * sizeof(long long)))) long long
_v2i64;
typedef __attribute__((__vector_size__(2 * sizeof(float)))) float _v2f32;
typedef __attribute__((__vector_size__(4 * sizeof(float)))) float _v4f32;
```

Таблица 2.2. Размер и выравнивание векторных типов данных DSP

Типы данных DSP	Размер в байтах	Размер в битах	Выравнивание в байтах
_v2i16,	4	32	4
_v4i16, _v2i32, _v2f32	8	64	8
_v8i16, _v4i32, _v2i64, _v4f32	16	128	16

3. ВЫРАВНИВАНИЕ СТЕКА

Указатель стека выравнивается по границе 8 байт (2 слова).

4. ИСПОЛЬЗОВАНИЕ РЕГИСТРОВ

Таблица 4.1.

Типы регистров	Назначение регистров
Регистры для передачи параметров	<p>Через регистры передается до трех переменных включительно соответственно размерности типа с учетом особенностей регистрового файла</p> <p>r0.s, r2.s, r4.s – для 16 разрядных типов r0.l, r2.l, r4.l – для 32 разрядных типов r0.d, r2.d, r4.d – для 64 разрядных типов r0.q, r2.q, r4.q – для 128 разрядных векторных типов</p> <p>Пример: void func(short a, int b, int c); a – r0.s, b – r2.l c – r4.l</p>
Регистры для возвращаемого значения	<p>соответственно размерности</p> <p>r0.s– для 16 разрядных типов r0.l– для 32 разрядных типов r0.d– для 64 разрядных типов r0.q– для 128 разрядных векторных типов</p> <p>Пример: long long func(); Возвращаемое значение в r0.d</p>
Регистры, сохраняемые вызываемой функцией (callee-saved registers)	r16.l, r17.l, r18.l, r19.l, r20.l, r21.l, r22.l, r23.l, r24.l, r25.l r17.d, r19.d, r21.d, r23.d, r25.d i3.l, i4.l, i5.l a3.l, a4.l, a5.l
Временные регистры, необходимые для компилятора. Можно использовать в ассемблерных вставках на протяжении вставки	r6.s, r7.s r6.l, r7.l r6.d, r7.d
Регистры для внешнего использования (например, ОС в обработчике прерываний или переключении задач)	r30.l, r31.l r30.d, r31.d
Указатель стека	a7.l
Указатель фрейма	a6.l
Зарезервированные регистры	R26.x – r29.x

5. ПОРЯДОК ВЫЗОВА ФУНКЦИИ

Рассмотрим программу (1):

```
int __attribute__((noinline))
f (int b)
{
    if (b != 1)
        return 1;
    return 0;
}

int main ()
{
    return f(1);
}
```

Скомпилируем её с уровнем оптимизации -O3 (в функции f атрибут noinline проставлен специально, чтобы компилятор не подставлял тело функции в точку вызова).

Рассмотрим получившийся ассемблерный код для DSP-ядра ELcore-30M:

```
00000004 <__dsp_f>:
00000004: 00400a21          clr1 r1.l
00000005: 00007935 00000001    cmpl 0x1, r0.l
00000007: 00039700 00000001    move.ne 0x1, r1.l
00000009: 00027951 00000001    andl 0x1, r1.l, r0.l
0000000b: f80001a0          rts

0000000c <__dsp_main>:
0000000c: 7ffff1ed          move 0xffffe, i7.s
0000000d: 000fe300          move (a7.l)+i0.l, r6.l
0000000e: 000d4d00          move ss.s, r6.s
0000000f: 07cfa267 00000001    move r6.l, (a7+0x00000001)
00000011: 0001ff00 00000001    move 0x1, r0.l
00000013: f80021ad          js 4 <__dsp_f>
00000014: 780011ed          move 0x2, i7.s
00000015: 000fe300          move (a7.l)+i0.l, r6.l
00000016: 7ffff1ed          move 0xffffe, i7.s
00000017: 00003780 00003f00    move (a7.l+i0.l), r6.d
00000019: 39800a5a          trl r7.l, r6.l
0000001a: 000d4c00          move r6.s, ss.s
0000001b: f80001a0          rts
```

Сейчас нас интересует передача аргумента в функцию f и исполнение функции f (pc=00000013).

Для передачи значения в функцию используется регистр r0.l:

```
00000011: 0001ff00 00000001    move 0x1, r0.l
```

5.1. Функции с определенным числом параметров

Расположение входных параметров подпрограммы в процедурах с определенным числом параметров известного размера:

- первые три входных параметра подпрограммы передаются через соответствующие типу регистры по указанному порядку:

для i8, i16 – r0.s, r2.s. r4.s;

для i32, f32, v2i16 - r0.l, r2.l. r4.l;

для i64, f64, v4i16, v2i32, v2f32 - r0.d, r2.d. r4.d;

для v2f64, v2i64, v4i32, v4f32, v4i32, v8i16 – r0.q, r2.q, r4.q;

для указателей - r0.l, r2.l, r4.l.

- остальные параметры, начиная с четвертого, передаются через стек.

- аргументы по значению(byVal), например, массивы, структуры и т.п., передаются через стек.

Для возврата значения из функции используется регистр r0.l:

```
20000010:    101de0e0          trl r2.l, r0.l
```

5.2. Передача возвращаемого значения

Возвращаемое значение передаётся

через регистр:

r0.s – для 16 разрядных типов

r0.l – для 32 разрядных типов

r0.d – для 64 разрядных типов

r0.q – для 128 разрядных векторных типов

через стек:

для аргументов по значению(byVal), например, массивы, структуры и т.п.

Подпрограмма вызывается инструкцией:

```
00000013:  f80021ad          js 4 <__dsp_f>
```

Выход из подпрограммы:

```
0000000b:  f80001a0          rts
```

5.3. Управление стеком

При вызове подпрограммы (callee) выделение памяти стека и возврат в исходное состояние стека осуществляется вызываемой подпрограммой (callee). Подготовка аргументов для подпрограммы (в том числе, расположенных на стеке) производится вызывающей программой (caller).

5.4. Вызов подпрограммы

Подпрограмма вызывается инструкцией `js`. Инструкция `js` записывает в аппаратный стек адрес возврата.

Сохранение адреса возврата аппаратного стека осуществляется вызываемой подпрограммой инструкцией (пролог):

ELcore-30M
<code>move ss, <ss_slot></code>

5.5. Выход из подпрограммы

Выход из подпрограммы осуществляется исполнением инструкции `rts`. Считывание значения адреса возврата аппаратного стека осуществляется вызываемой подпрограммой инструкцией (эпилог):

ELcore-30M
<code>move <ss_slot>, ss</code>

5.6. Указатель фрейма

Регистр `a7` - stack pointer (`sp`). Регистр `a6` - frame pointer (`fp`).

Указатель фрейма `FP` необходим в следующих случаях:

- `debug/unwind` при компиляции для целей отладки с `-O0/-g`, при обработке `c++` исключений, т.е. в тех случаях, когда требуется раскрутка стека;
- когда указатель стека (`sp`) смещается динамически в процессе исполнения подпрограммы (например, используется `malloc`). В этом случае привязка к `sp` невозможна и нужен `fp`;
- для функций с переменным числом аргументов;

В остальных случаях можно использовать `sp`. Использование только указателя стека является оптимизацией (можно было бы оставить использование `fp` всегда, просто в некоторых случаях это не является необходимым).

Обратите внимание, что `caller` не знает о том, что будет использовать `callee` и, в общем случае, они могут компилироваться независимо с разными ключами компиляции. Поэтому `sp+spOffset` (если можно использовать `sp`) и `fp+fpOffset` должны равняться одному и тому же адресу памяти, что и гарантируется компилятором.

5.7. Слот для указателя фрейма

В случаях, когда используется `fp(4.6)`, для его сохранения/восстановления необходим `fp` слот. Если в `callee` имеются вызовы подпрограмм, для сохранения/восстановления адреса возврата при входе/выходе в/из `callee` необходим `ss` слот.

В остальных случаях слоты не используются, а `sp` при необходимости смещается и восстанавливается соответственно.

5.8. Организация стека

Направление роста стека – вверх (от больших адресов памяти к меньшим адресам памяти).

Порядок передачи параметров через стек – прямой (у параметра с меньшим порядком меньший адрес в стеке).

Указатель стека выравнивается по границе 8 байт (2 слова). Для любого параметра на стеке выделяется минимально 8 байт. Если размер параметра больше, выделяется память на стеке с учётом выравнивания на 8 байт.

5.9. Callee-saved registers (CSR)

CSR - это регистры, которые "не меняются" вызовами, т.е. `callee` гарантирует их сохранность:

`r16.l, r17.l, r18.l, r19.l, r20.l, r21.l, r22.l, r23.l, r24.l, r25.l`

`r17.d, r19.d, r21.d, r23.d, r25.d`

`r17.q, r19.q, r21.q, r23.q, r25.q`

`i3.l, i4.l, i5.l`

`a3.l, a4.l, a5.l`

5.10. Функции с переменным числом параметров

При вызове функции с переменным числом параметром могут передаваться явно заданные параметры (далее, обязательные) и неявно заданные, которые определяют переменное число параметров (далее, необязательные).

Передача аргументов в функцию с переменным числом параметров осуществляется следующим образом:

- обязательные параметры передаются в соответствии с правилами п.4.1 «Функции с определённым числом параметров»;
- необязательные параметры передаются в соответствии с правилами п.4.1 «Функции с определённым числом параметров»;

- для поддержки макросов из `<stdarg.h>` callee (всегда, независимо от того используется ли `<stdarg.h>`) формирует однородный `va_list`. Формирование `va_list` осуществляется выделением памяти (под каждый параметр выделяется 8 байт) и копированием необязательных параметров из незанятых постоянными параметрами регистров в память. Данные переносятся таким образом, чтобы получить непрерывный `va_list` (т.е. занимающий область в памяти без разрывов);
- `va_list` находится на наибольших адресах в памяти относительно других параметров, т.к. порядок передачи аргументов прямой, а переменные аргументы всегда передаются последними.

6. СТРУКТУРА ФРЕЙМА. СТРУКТУРА ПРОЛОГА И ЭПИЛОГА

6.1. Функция с фиксированным числом аргументов без слотов для FP

Функция с фиксированным числом параметров, слоты (4.7) не нужны.

Структура фрейма:

```

---- ---- sp_new(a7) = sp - (sizeof(objects) + sizeof(CSR))
objects: (4.3)
локальные переменные;
память для, используемая в процессе вычислений;
---- ----
callee-saved
registers (CSR); (4.8)
---- ---- sp(a7)
fixedObjects:
аргументы, передаваемые через память
и byVal параметры; (4.1)
Структура пролога:
sub (sizeof(objects) + sizeof(CSR)), a7, a7;
сохранение CSR;
Структура эпилога:
восстановление CSR;
add (sizeof(objects) + sizeof(CSR)), a7, a7;

```

6.2. Функция с фиксированным числом аргументов без слотов для FP

Обычная функция с фиксированным числом параметров, слоты (4.7) нужны.

Структура фрейма:

```

---- ---- sp_new(a7) = sp - (sizeof(objects) + sizeof(CSR) + 2) (слоты)
objects: (4.3)
локальные переменные;
память используемая в процессе вычислений;
---- ----
callee-saved registers (CSR); (4.8)
---- ---- fp(a6)
слоты: fp-слот (4 байта) (4.7)
----
ss-слот (4 байта)
---- ---- sp(a7)
fixedObjects:
аргументы передаваемые через память
и byVal параметры; (4.1)

```

Структура пролога:

ELcore-30M:

```

смещение sp:
    move -(sizeof(objects) + sizeof(CSR) + 2(слоты)), i7.s
    move (a7.l)+i7.l, r6.l
сохранение ss:
    move ss.s, r6.s
    move r6.l, (a7+(sizeof(objects) + sizeof(CSR) + 1))
сохранение fp:
    move a6.s, r6.s
    move r6.l, (a7+(sizeof(objects) + sizeof(CSR)))
смещение fp:
    move a7.s, r6.s
    add (sizeof(objects) + sizeof(CSR)), r6.s
    move r6.s, a6.s
сохранение CSR;

```

Структура эпилога:

ELcore-30M:

```

восстановление CSR;
восстановление sp:
    move a6.s, r6.s
    add 0x2, r6.s
    move r6.s, a7.s
восстановление fp (читаем сразу два слота):
    move (a6.l), r6.d
    move r6.s, a6.s
восстановление ss:
    trl r7.l, r6.l
    move r6.s, ss.s
возврат;
    rts

```

6.3. Функция с переменным числом параметров

Такая функция всегда использует fp слот (4.6).

Структура фрейма:

```

---- ---- sp_new(a7) = sp - (sizeof(objects) + sizeof(CSR) + 2 (слоты) +
sizeof(регистры в va_list) (4.9))
objects: (4.3)
локальные переменные;
память используемая в процессе вычислений;
---- ----
callee-saved
registers (CSR); (4.8)
---- ---- fp(a6)
слоты: fp-слот (4 байта) (4.7)
----
ss-слот (4 байта)
---- ---- sp(a7)
fixedObjects:
аргументы передаваемые через память
и byVal параметры; (4.1)
----

```

va_list (4.9)

Структура пролога:

ELcore-30M:

выделяем память под регистры в va_list:

```

    move -sizeof(регистры в va_list)(4.9), i7.s
    move (a7.l)+i7.l, r6.l
смещение sp:
    move -(sizeof(objects) + sizeof(CSR) + 2(слоты)), i7.s
    move (a7.l)+i7.l, r6.l
сохранение ss:
    move ss.s, r6.s
    move r6.l, (a7+(sizeof(objects) + sizeof(CSR) + 1))
сохранение fp:
    move a6.s, r6.s
    move r6.l, (a7+(sizeof(objects) + sizeof(CSR)))
смещение fp:
    move a7.s, r6.s
    add (sizeof(objects) + sizeof(CSR)), r6.s
    move r6.s, a6.s
сохранение CSR;
```

Структура эпилога:

ELcore-30M:

восстановление CSR;

```

восстановление sp:
    move a6.s, r6.s
    add 0x2, r6.s
    move r6.s, a7.s
восстановление fp (читаем сразу два слота):
    move (a6.l), r6.d
    move r6.s, a6.s
восстановление ss:
    trl r7.l, r6.l
    move r6.s, ss.s
освобождаем память под регистры в va_list:
    move sizeof(регистры в va_list)(4.9), i7.s
    move (a7.l)+i7.l, r6.l
возврат;
    rts
```

ОБОЗНАЧЕНИЯ

callee-saved registers (CSR) – регистры, сохраняемые вызываемой подпрограммой

callee – вызываемая подпрограмма

caller – вызывающая подпрограмма

ПРИМЕР 1. ELCORE-30M. УРОВЕНЬ ОПТИМИЗАЦИИ -O0.

Скомпилируем программу (1) с уровнем оптимизации -O0 для DSP-ядра ELCore-30M.

```
int __attribute__((noinline))
f (int b)
{
    if (b != 1)
        return 1;
    return 0;
}

int main ()
{
    return f(1);
}
```

Листинг дизассемблера

```
00000000 <_dsp_f>:
00000000: 7fffd1ed          move 0xfffa, i7.s
00000001: 000fe300          move (a7.l)+i0.s, r6.l
00000002: 000c3500          move a6.s, r6.s
00000003: 07cfa267 00000004 move r6.l, (a7.l+0x00000004)
00000005: 000c3d00          move a7.s, r6.s
00000006: 30002184          add 0x4, r6.s
00000007: 000c3400          move r6.s, a6.s
00000008: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
0000000a: 00400a5a          trl r0.l, r1.l
0000000b: 00007935 00000001      cmpl 0x1, r0.l
0000000d: 09800a5a          trl r1.l, r6.l
0000000e: 07cf2267 ffffffff          move r6.l, (a6.l+0xffffffff)
00000010: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
00000012: 3800399c          b.eq 0x7 <0x19>
00000013: f800099c          b 0x1 <0x14>
00000014: 0001ff00 00000001      move 0x1, r0.l
00000016: f800399e          bd 0x7 <0x1d>
00000017: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
00000019: 00000a21          clrl r0.l
0000001a: f800199e          bd 0x3 <0x1d>
0000001b: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
0000001d: 07c32367 ffffffff          move (a6.l+0xffffffff), r0.l
0000001f: 000c3500          move a6.s, r6.s
00000020: 30001184          add 0x2, r6.s
00000021: 000c3c00          move r6.s, a7.s
00000022: 000f1300          move (a6.l), r6.l
00000023: 000c3400          move r6.s, a6.s
00000024: f80001a0          rts

00000025 <_dsp_main>:
00000025: 7fffe1ed          move 0xfffc, i7.s
00000026: 000fe300          move (a7.l)+i0.s, r6.l
00000027: 000d4d00          move ss.s, r6.s
00000028: 07cfa267 00000003      move r6.l, (a7.l+0x00000003)
0000002a: 000c3500          move a6.s, r6.s
0000002b: 07cfa267 00000002      move r6.l, (a7.l+0x00000002)
0000002d: 000c3d00          move a7.s, r6.s
```

```

0000002e: 30001184          add 0x2, r6.s
0000002f: 000c3400          move r6.s, a6.s
00000030: 00000a21          clrl r0.l
00000031: 07c32267 ffffffff        move r0.l, (a6.l+0xffffffff)
00000033: 0001ff00 00000001    move 0x1, r0.l
00000035: f80001ad          js 0 <__dsp_f>
00000036: 000c3500          move a6.s, r6.s
00000037: 30001184          add 0x2, r6.s
00000038: 000c3c00          move r6.s, a7.s
00000039: 00003780 00001d00    move (a6.l), r6.d
0000003b: 000c3400          move r6.s, a6.s
0000003c: 39800a5a          trl r7.l, r6.l
0000003d: 000d4c00          move r6.s, ss.s
0000003e: f80001a0          rts

```

Выделяется четыре слова под четыре регистра и два под слоты.

Соответственно в прологе выполняются следующие действия:

Сдвигаем стек a7.l[00007ffc --> 00007ff6]:

```

dsp0 pc 00000025 step 00000000 fmt3 7fffeled :
00000025 move fffc, i7.s[0000 --> fffc],

dsp0 pc 00000026 step 00000001 fmt4 000fe300 :
00000026 nop
00000026 move (a7.l)+i7.s[00007ffc --> 00007ff8],
r6.l[cdcdcdcd --> cdcdcdcd],

```

Сохраняем аб в слот:

```

dsp0 pc 00000027 step 00000002 fmt6 000d4d00 :
00000027 nop
00000027 move ss.s[0000], r6.s[cdcd --> 005f],

dsp0 pc 00000028 step 00000003 fmt7t 07cfa267 00000003 :
00000028 move r6.l[cdcd005f], (a7.l+00000003)[00007ff8 -->
00007ff8],

dsp0 pc 0000002a step 00000004 fmt6 000c3500 :
0000002a nop
0000002a move a6.l[00007ffc], r6.s[005f --> 7ffc],

dsp0 pc 0000002b step 00000005 fmt7t 07cfa267 00000002 :
0000002b move r6.l[cdcd7ffc], (a7.l+00000002)[00007ff8 -->
00007ff8],

```

В данном примере на эпилоге для ELcore-30M выполнены следующие действия:

Восстановление a7:

```

dsp0 pc 00000036 step 00000040 fmt6 000c3500 :
00000036 nop
00000036 move a6.l[00007ffa], r6.s[7ffa --> 7ffa],

dsp0 pc 00000037 step 00000041 fmt3 30001184 :
00000037 add 0002, r6.s[7ffa], r6.s[7ffa --> 7ffc],

dsp0 pc 00000038 step 00000042 fmt6 000c3c00 :
00000038 nop

```

00000038	move	r6.s[7ffc], a7.l[00007ff8 --> 00007ffc],
----------	------	--

Восстановление а6:

dsp0 pc	00000039	step	00000043	fmt9a	00003780	00001d00	:
	00000039	nop					
	00000039	nop					
	00000039	move		(a6.l)	[00007ffa --> 00007ffa],	r6.d[cdcdcdcd	
				cdcd7ffc --> cdcd005f	cdcd7ffc],		
dsp0 pc	0000003b	step	00000044	fmt6	000c3400		:
	0000003b	nop					
	0000003b	move		r6.s[7ffc],	a6.l[00007ffa --> 00007ffc],		

Возврат:

dsp0 pc	0000003e	step	00000047	fmt3m	f80001a0	:
	0000003e	rts		pc[0000003e --> 0000005f],		

Так как нет вызовов функции внутри f, адрес возврата с аппаратного стека не сохраняется и не восстанавливается через ss слот.

Возврат из функции производится автоматически инструкцией rts, которая считывает адрес возврата с аппаратного стека.

ПРИМЕР 2. ELCORE-30M. ОРГАНИЗАЦИЯ СТЕКА.

```
#define N 0x100

int g1[N], g2[N], g3[N], g4[N];

int __attribute__((noinline))
f (int n, int p1, int p2, int p3, int p4)
{
    if (n >= N) {
        int i;
        int l1[N], l2[N], l3[N], l4[N];
        int v1, v2, v3, v4;
        v1 = v2 = v3 = v4 = 0;
        for (i = 0; i < N; i++) {l1[i] = i; if (i == p1) v1 = l1[i];}
        for (i = 0; i < N; i++) {l2[i] = N - i; if (i == p2) v2 = l2[i];}
        for (i = 0; i < N; i++) {l3[i] = i << 1; if (i == p3) v3 = l3[i];}
        for (i = 0; i < N; i++) {l4[i] = (N - i) << 1; if (i == p4) v4 =
14[i];}
        for (i = 0; i < N; i++) g1[i] = l1[i] + l2[i] + v1 * v2 / v3;
        for (i = 0; i < N; i++) g2[i] = l1[i] - l2[i] + v2 * v3 / v4;
        for (i = 0; i < N; i++) g3[i] = l3[i] + l4[i] + v3 * v4 / v1;
        for (i = 0; i < N; i++) g4[i] = l3[i] - l4[i] + v4 * v1 / v2;
    }
    return 0;
}

int main ()
{
    return f(N, 1, 2, 3, 4);
}
```

В программе компилятор выполняет деление с помощью библиотечной функции `__divsi3`. Это приводит к тому, что в `f` имеются вложенные вызовы и необходимо использовать `ss` слот.

Рассмотрим получившийся ассемблерный код для ELcore-30M:

```

20000000 <_f>:
00000000: 7fdfc1ed          move 0xfbf8, i7.s
00000001: 000fe300          move (a7.l)+i0.s, r6.l
00000002: 000d4d00          move ss.s, r6.s
00000003: 07cfa267 00000407 move r6.l, (a7.l+0x00000407)
00000005: 000c3500          move a6.s, r6.s
00000006: 07cfa267 00000406 move r6.l, (a7.l+0x00000406)
00000008: 000c3d00          move a7.s, r6.s
00000009: 30203184          add 0x406, r6.s
0000000a: 000c3400          move r6.s, a6.s
0000000b: 0f8c1d67          move a3.l, r6.l
0000000c: 07cf2267 ffffffff move r6.l, (a6.l+0xffffffff)
0000000e: 07e32267 ffffffff move r16.l, (a6.l+0xffffffff)
00000010: 89800a5a          trl r17.l, r6.l
00000011: 07cf2267 ffffffff move r6.l, (a6.l+0xffffffff)
00000013: 07e72267 ffffffff move r18.l, (a6.l+0xffffffff)
00000015: 99800a5a          trl r19.l, r6.l
00000016: 07cf2267 ffffffff move r6.l, (a6.l+0xffffffff)
....
00000071: f80b69ad          js 16d <_dsp_divsi3>
....
0000010d: 07e32367 ffffffff move (a6.l+0xffffffff), r16.l
0000010f: 07cf2367 ffffffff move (a6.l+0xffffffff), r6.l
00000111: 34400a5a          trl r6.l, r17.l
00000112: 07e72367 ffffffff move (a6.l+0xffffffff), r18.l
00000114: 07cf2367 ffffffff move (a6.l+0xffffffff), r6.l
00000116: 34c00a5a          trl r6.l, r19.l
00000117: 000c3500          move a6.s, r6.s
00000118: 30001184          add 0x2, r6.s
00000119: 000c3c00          move r6.s, a7.s
0000011a: 00003780 00001d00 move (a6.l), r6.d
0000011c: 000c3400          move r6.s, a6.s
0000011d: 39800a5a          trl r7.l, r6.l
0000011e: 000d4c00          move r6.s, ss.s

```

- используется смещение регистра a7 на значение размера стека
- используется сохранение и восстановление значения регистра a6.s
- осуществляется сохранение/восстановление callee-saved registers r16.l, r17.l, r18.l (CSR);
- обращение к стеку осуществляется с помощью a7(sp), т.к. в соответствии с условиями 4.6. fp всё ещё не нужен;
- простейший способ получить пример доступа к стеку при помощи fp (в соответствии с 4.6.)
- скомпилировать например (2a) с -O0/-g;

ПРИМЕР 3. ИСПОЛЬЗОВАНИЕ CALLEE-SEVED REGISTERS.

Изменим программу (2а), а именно заменим "l[4*N]" на "l1[n], l2[n], l3[n], l4[n]".

Получим программу (2b):

```
#define N 0x100

int g1[N], g2[N], g3[N], g4[N];

int __attribute__((noinline))
f (int n, int p1, int p2, int p3, int p4)
{
    if (n >= N) {
        int i;
        int l1[n], l2[n], l3[n], l4[n];
        int v1, v2, v3, v4;
        v1 = v2 = v3 = v4 = 0;
        for (i = 0; i < N; i++) {l1[i] = i; if (i == p1) v1 = l1[i];}
        for (i = 0; i < N; i++) {l2[i] = N - i; if (i == p2) v2 = l2[i];}
        for (i = 0; i < N; i++) {l3[i] = i << 1; if (i == p3) v3 = l3[i];}
        for (i = 0; i < N; i++) {l4[i] = (N - i) << 1; if (i == p4) v4 =
l4[i];}
        for (i = 0; i < N; i++) g1[i] = l1[i] + l2[i] + v1 * v2 / v3;
        for (i = 0; i < N; i++) g2[i] = l1[i] - l2[i] + v2 * v3 / v4;
        for (i = 0; i < N; i++) g3[i] = l3[i] + l4[i] + v3 * v4 / v1;
        for (i = 0; i < N; i++) g4[i] = l3[i] - l4[i] + v4 * v1 / v2;
    }
    return 0;
}

int main ()
{
    return f(N, 1, 2, 3, 4);
}
```

Скомпилируем с опцией оптимизации -O3.

Рассмотрим получившийся ассемблерный код для ELcore-30M:

```

00000000 <__dsp_f>:
....
0000000b: 0f8c1d67                move a3.l, r6.l
0000000c: 07cf2267 ffffffff             move r6.l, (a6.l+0xffffffff)
0000000e: 07e32267 ffffffff             move r16.l, (a6.l+0xffffffffe)
00000010: 89800a5a                trl r17.l, r6.l
00000011: 07cf2267 ffffffff             move r6.l, (a6.l+0xffffffffd)
00000013: 07e72267 ffffffff             move r18.l, (a6.l+0xffffffffc)
00000015: 99800a5a                trl r19.l, r6.l
00000016: 07cf2267 ffffffff             move r6.l, (a6.l+0xffffffffb)
00000018: 07eb2267 ffffffff             move r20.l, (a6.l+0xffffffffa)
0000001a: a9800a5a                trl r21.l, r6.l
0000001b: 07cf2267 ffffffff             move r6.l, (a6.l+0xffffffff9)
0000001d: 07ef2267 ffffffff             move r22.l, (a6.l+0xffffffff8)
0000001f: b9800a5a                trl r23.l, r6.l
00000020: 07cf2267 ffffffff             move r6.l, (a6.l+0xffffffff7)
00000022: 07f32267 ffffffff             move r24.l, (a6.l+0xffffffff6)
....
00000109: 07cf2367 ffffffff             move (a6.l+0xffffffff), r6.l
0000010b: 0f8c1c67                move r6.l, a3.l
0000010c: 07e32367 ffffffff             move (a6.l+0xffffffffe), r16.l
0000010e: 07cf2367 ffffffff             move (a6.l+0xffffffffd), r6.l
00000110: 34400a5a                trl r6.l, r17.l
00000111: 07e72367 ffffffff             move (a6.l+0xffffffffc), r18.l
00000113: 07cf2367 ffffffff             move (a6.l+0xffffffffb), r6.l
00000115: 34c00a5a                trl r6.l, r19.l
00000116: 07eb2367 ffffffff             move (a6.l+0xffffffffa), r20.l
00000118: 07cf2367 ffffffff             move (a6.l+0xffffffff9), r6.l
0000011a: 35400a5a                trl r6.l, r21.l
0000011b: 07ef2367 ffffffff             move (a6.l+0xffffffff8), r22.l
0000011d: 07cf2367 ffffffff             move (a6.l+0xffffffff7), r6.l
0000011f: 35c00a5a                trl r6.l, r23.l
00000120: 07f32367 ffffffff             move (a6.l+0xffffffff6), r24.l
....

```

Как видно из листинга дизассемблера инструкции с `pc=0x0b-0x22` сохраняют регистры `a3.l`, `r17.l-r24.l`, а инструкции с `pc = 0x109-0x120` восстанавливают эти значения. Таким образом, обеспечивается сохранение значений регистров вызывающей функции.

ПРИМЕР 4. ФУНКЦИЯ С ПЕРЕМЕННЫМ ЧИСЛОМ ПАРАМЕТРОВ.

Рассмотрим пример(3) функции с переменным числом параметров:

```
#include <stdarg.h>

typedef struct {
    char a[9];
} big;

int
f (big x, int b, ...)
{
    int arg;
    va_list ap;

    if (x.a[0] != 'a')
        return 1;
    if (b != 0x111)
        return 0x111;
    va_start (ap, b);
    arg = va_arg (ap, int);
    va_end (ap);
    if (arg != 0x222)
        return 0x222;
    return 0;
}

int main ()
{
    static big x = { "a" };

    return f (x, 0x111, 0x222, x);
}
```

Скомпилируем программу с опцией оптимизации -O3. Рассмотрим получившийся ассемблерный код для ELcore-30M:

```

00000000 <__dsp_f>:
00000000: 7fffe1ed          move 0xfffc, i7.s
00000001: 000fe300          move (a7.l)+i0.s, r6.l
00000002: 7fffb1ed          move 0xffff6, i7.s
00000003: 000fe300          move (a7.l)+i0.s, r6.l
00000004: 000d4d00          move ss.s, r6.s
00000005: 07cfa267 00000009 move r6.l, (a7.l+0x00000009)
00000007: 000c3500          move a6.s, r6.s
00000008: 07cfa267 00000008 move r6.l, (a7.l+0x00000008)
0000000a: 000c3d00          move a7.s, r6.s
0000000b: 30004184          add 0x8, r6.s
0000000c: 000c3400          move r6.s, a6.s
0000000d: 07e32267 ffffffff          move r16.l, (a6.l+0xffffffff)
0000000f: 07e72267 ffffffff          move r18.l, (a6.l+0xffffffff)
00000011: 77ffc1ed          move 0xffff8, i6.s
00000012: 00002780 00003c00 move r4.d, (a6.l+i0.s)
00000014: 77ffd1ed          move 0xffffa, i6.s
00000015: 00001780 00003c00 move r2.d, (a6.l+i0.s)
00000017: 04000a5a          trl r0.l, r16.l
00000018: 0f803567          move a6.l, r0.l
00000019: 100178e8          lsl 0x2, r0.l, r0.l
0000001a: 0f843567          move a6.l, r2.l
0000001b: 108578e8          lsl 0x2, r2.l, r2.l
0000001c: 00847924 00000008 addl 0x8, r2.l, r2.l
0000001e: 118578fc          asr 0x2, r2.l, r6.l
0000001f: 000c0400          move r6.s, a0.s
00000020: 0f820567          move a0.l, r1.l
00000021: 104378e8          lsl 0x2, r1.l, r1.l
00000022: 148378fc          asr 0x2, r1.l, r18.l
00000023: 00807924 00000018 addl 0x18, r0.l, r2.l
00000025: 00007924 00000008 addl 0x8, r0.l, r0.l
00000027: 0009ff00 00000010 move 0x10, r4.l
00000029: f805f9ad          js bf <__dsp_memmove>
0000002a: 77ffc1ed          move 0xffff8, i6.s
0000002b: 00000780 00003d00 move (a6.l+i0.s), r0.d
0000002d: 700041ed          move 0x8, i6.s
0000002e: 00000780 00003c00 move r0.d, (a6.l+i0.s)
00000030: 77ffd1ed          move 0xffffa, i6.s
00000031: 00000780 00003d00 move (a6.l+i0.s), r0.d
00000033: 700031ed          move 0x6, i6.s
00000034: 00000780 00003c00 move r0.d, (a6.l+i0.s)
00000036: 00240400          move r18.s, a0.s
00000037: 00001300          move (a0.l), r0.l
00000038: 00007951 000000ff andl 0xff, r0.l, r0.l
0000003a: 00030995          cmp 0x61, r0.s
0000003b: 2800b99c          b.ne 0x17 <0x52>
0000003c: 04207935 00000111 cmpl 0x111, r16.l
0000003e: 2800b99c          b.ne 0x17 <0x55>
0000003f: 0f803567          move a6.l, r0.l
00000040: 100178e8          lsl 0x2, r0.l, r0.l
00000041: 00007924 00000018 addl 0x18, r0.l, r0.l
00000043: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
00000045: 00007924 00000008 addl 0x8, r0.l, r0.l
00000047: 07c32267 ffffffff          move r0.l, (a6.l+0xffffffff)
00000049: 0001ff00 00000222 move 0x222, r0.l
0000004b: 07c72367 00000006 move (a6.l+0x00000006), r2.l

```

```

0000004d: 00847935 00000222    cml 0x222, r2.l
0000004f: 00019f00 00000000    move.eq 0x0, r0.l
00000051: f800319c      b 0x6 <0x57>
00000052: f800299e      bd 0x5 <0x57>
00000053: 0001ff00 00000001    move 0x1, r0.l
00000055: 0001ff00 00000111    move 0x111, r0.l
00000057: 07e32367 ffffffff    move (a6.l+0xffffffff), r16.l
00000059: 07e72367 ffffffff    move (a6.l+0xffffffffe), r18.l
0000005b: 000c3500      move a6.s, r6.s
0000005c: 30001184      add 0x2, r6.s
0000005d: 000c3c00      move r6.s, a7.s
0000005e: 00003780 00001d00    move (a6.l), r6.d
00000060: 000c3400      move r6.s, a6.s
00000061: 39800a5a      trl r7.l, r6.l
00000062: 000d4c00      move r6.s, ss.s
00000063: 780021ed      move 0x4, i7.s
00000064: 000fe300      move (a7.l)+i0.s, r6.l
00000065: f80001a0      rts

00000066 <__dsp_main>:
00000066: 7fff91ed      move 0xffff2, i7.s
00000067: 000fe300      move (a7.l)+i0.s, r6.l
00000068: 000d4d00      move ss.s, r6.s
00000069: 07cfa267 0000000d    move r6.l, (a7.l+0x0000000d)
0000006b: 000c3500      move a6.s, r6.s
0000006c: 07cfa267 0000000c    move r6.l, (a7.l+0x0000000c)
0000006e: 000c3d00      move a7.s, r6.s
0000006f: 30006184      add 0xc, r6.s
00000070: 000c3400      move r6.s, a6.s
00000071: 07e32267 ffffffff    move r16.l, (a6.l+0xffffffff)
00000073: 89800a5a      trl r17.l, r6.l
00000074: 07cf2267 ffffffff    move r6.l, (a6.l+0xffffffffe)
00000076: 07e72267 ffffffff    move r18.l, (a6.l+0xfffffffffd)
00000078: 00007f00 00000000    move 0x0, a0.s
0000007a: 0f803d67      move a7.l, r0.l
0000007b: 100178e8      lsl 0x2, r0.l, r0.l
0000007c: 04007924 00000010    addl 0x10, r0.l, r16.l
0000007e: 0f820567      move a0.l, r1.l
0000007f: 104378e8      lsl 0x2, r1.l, r1.l
00000080: 144378e8      lsl 0x2, r1.l, r17.l
00000081: 0025ff00 00000009    move 0x9, r18.l
00000083: 88800a5a      trl r17.l, r2.l
00000084: 91000a5a      trl r18.l, r4.l
00000085: f808f1ad      js 11e <__dsp_memcpy>
00000086: 80000a5a      trl r16.l, r0.l
00000087: 88800a5a      trl r17.l, r2.l
00000088: 91000a5a      trl r18.l, r4.l
00000089: f808f1ad      js 11e <__dsp_memcpy>
0000008a: 0005ff00 00000222    move 0x222, r2.l
0000008c: 0001ff00 00000111    move 0x111, r0.l
0000008e: f80001ad      js 0 <__dsp_f>
0000008f: 07e32367 ffffffff    move (a6.l+0xffffffff), r16.l
00000091: 07cf2367 ffffffff    move (a6.l+0xffffffffe), r6.l
00000093: 34400a5a      trl r6.l, r17.l
00000094: 07e72367 ffffffff    move (a6.l+0xfffffffffd), r18.l
00000096: 000c3500      move a6.s, r6.s
00000097: 30001184      add 0x2, r6.s

```

```
00000098: 000c3c00          move r6.s, a7.s
00000099: 00003780 00001d00  move (a6.l), r6.d
0000009b: 000c3400          move r6.s, a6.s
0000009c: 39800a5a          trl r7.l, r6.l
0000009d: 000d4c00          move r6.s, ss.s
0000009e: f80001a0          rts
```

- caller передаёт необязательные аргументы в соответствии с 4.1, а не через стек по причине того, что в языке C допускается использование функций без предварительной декларации;
- функция без предварительной декларации считается функцией с переменным числом аргументов. Строго говоря, такая ситуация является ошибкой программирования.
- копирование аргументов по значению(byVal) здесь осуществляется с помощью __dsp_memcpy:

ИСТОРИЯ ИЗМЕНЕНИЙ

Версия	До изменения	После изменения
REV 1.01	Раздел 3. Указатель стека – а6.1 Указатель фрейма – а7.1	Раздел 3. Указатель стека – а7.1 Указатель фрейма – а6.1
		Раздел 6. Обновлены примеры для структуры пролога и эпилога
REV 1.02	Разделы 4, 5, 6, 7	Разделы 4, 5 обновлены и дополнены примерами, исправлена опечатка в определении callee-saved registers
REV 1.03	Весь документ	Переписан. Добавлены примеры с объяснением. Изменение оформления
REV 2.01	Весь документ	Версия для setup