Есть ли у вас вопросы?

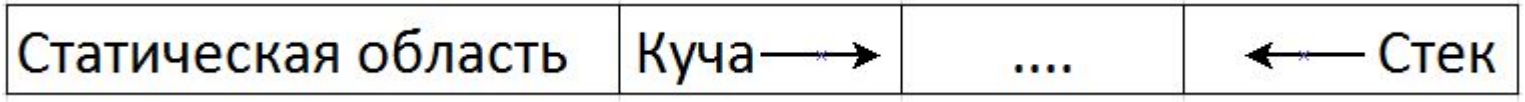
Краткое содержание предыдущей серии

- Как в ассемблере происходит сравнение?
- Как используется результат сравнения?
- В чем отличие логических операций от битовых?
- Какие вы помните логические операции?
- А битовые?
- Чем опасны сдвиги в C?

Краткое содержание этой серии

- Модель памяти в языке С
- Функции в языке С
- Функции в ассемблере
- Еще о командах перехода

Модель оперативной памяти в языке С



- Статическая область – ее размер известен при компиляции; там хранятся глобальные и статические (static) переменные, там могу хранится константы.

- Куча – область, из которой выделяется динамическая память (malloc() в С, new в С++). Ее максимальный размер задается *как-то (например, программист просто выбирает число).*

- Стек – его размер меняется при работе программы. Максимальный размер задается *как-то.*

Плохие ситуации: выход за границы стека, выход за границы кучи, встреча кучи и стека.

Функции в языке C: объявление

Объявление (прототип) – declaration:

Что такое объявление?

Это «обещание», что где-то написано тело функции.

Пример: char foo(int a);

Где должно располагаться объявление?

- До вызова. Т.е. выше по тексту.
- В заголовочном файле (.h), если это глобальная функция.
- В том же файле .c, если это функция static.

Объявление может быть совмещено с телом функции.

Ошибки линкера «undefined symbol имяФункции» означают, что есть объявление, но нет тела.

Функции в языке C: объявление

```
void * foo(int a);
```

- void * - тип возвращаемого значения
- foo – имя функции
- int a – тип и имя параметра (аргумента)
  - аргументов может быть много, они разделяются запятой
  - аргументов может быть переменное количество
- ; - обязательный элемент синтаксиса, если дальше нет тела функции

Функции в языке C: определение

Что такое определение (тело) – definition?

Это сам код функции, который будет выполняться при вызове.

```
char foo(int a)
{
    ...
}
```

Функции в языке C: вызов

Как происходит вызов функции:

```
char foo(int a); //определение должно быть до вызова
```

char result = foo(2);

- 2 – параметр, передаваемый в функцию
- result – переменная, в которую запишется возвращаемое значение

Функции в языке C: вызов

```
char foo(int a);
...
char result = foo(2);
```

После этой строки управление «мистическим» образом передается на первую строку тела функции, причем параметр a будет равным 2.

Параметры функции внутри нее – просто локальные переменные.

Функции и процедуры

В настоящее время различие минимально:

процедуры не возвращают значение,

а функции – могут возвращать (но не обязательно).

Функции в ассемблере

Вызов функции в ассемблере: команды

- BL address
- BLX register

Переход с сохранением адреса возврата в регистре R14 (Link Register, LR).

Адрес возврата – адрес следующей команды после BL.

А по какому адресу нужно перейти, чтобы попасть в функцию?

По адресу первой инструкции в ее теле.

Функции в ассемблере

Что нужно сделать, чтобы вызвать функцию?

- Как-то передать параметры
- Как-то передать управление
- Как-то вернуться к месту вызова
- Как-то вернуть значение

При этом:

- Внутри функции тоже могут вызвать функцию
- Может быть даже ту же самую (рекурсия)
- Ничего не должно сломаться!

Функции в ассемблере: что может сломаться?

Весь код в ассемблере использует регистры.

Код в функции тоже использует регистры.

```
int a = 1 + sin(3.14);
MOV r0, 1 ; собираюсь складывать 1 и синус
(вызов sin) ; вызываю sin
.. а если функция sin тоже использовала r0?
```

Что же делать?

Функции в ассемблере: что может сломаться?

Содержимое регистров может быть испорчено при вызове функций. Что делать?

Содержимое регистров нужно куда-то сохранять до вызова и восстанавливать после.

Куда сохранять?

Функции в ассемблере: состояние регистров

Куда сохранять состояние регистров?

- В специальную статическую область памяти (архитектура 8051)
- Аппаратно в теневые регистры
- **В стек**

Число регистров неизменно – всегда известно, сколько памяти нужно для сохранения.

Функции в ассемблере

А не нужно ли сохранять что-нибудь еще?

- Состояние LR для текущей функции (но это регистр)
- Локальные переменные текущей функции?

Кстати, а где хранятся локальные переменные?

Локальные переменные хранятся:

- В регистрах
- В специальной статической области памяти
- **В стеке**

Поэтому их не надо сохранять, но нужно не задеть случайно.

Функции в ассемблере

А как передавать параметры?

- На регистрах (их ведь все равно сохраняем)
- В специальной статической области памяти
- В куче
- **В стеке**

А как возвращать значение?

См. выше

Стек

Доступ к стеку:

- спец. команды push и pop
- через SP (регистр – указатель стека)
- или через еще какой-нибудь регистр

Т.е. к стеку можно обращаться через косвенно-регистровую адресацию.

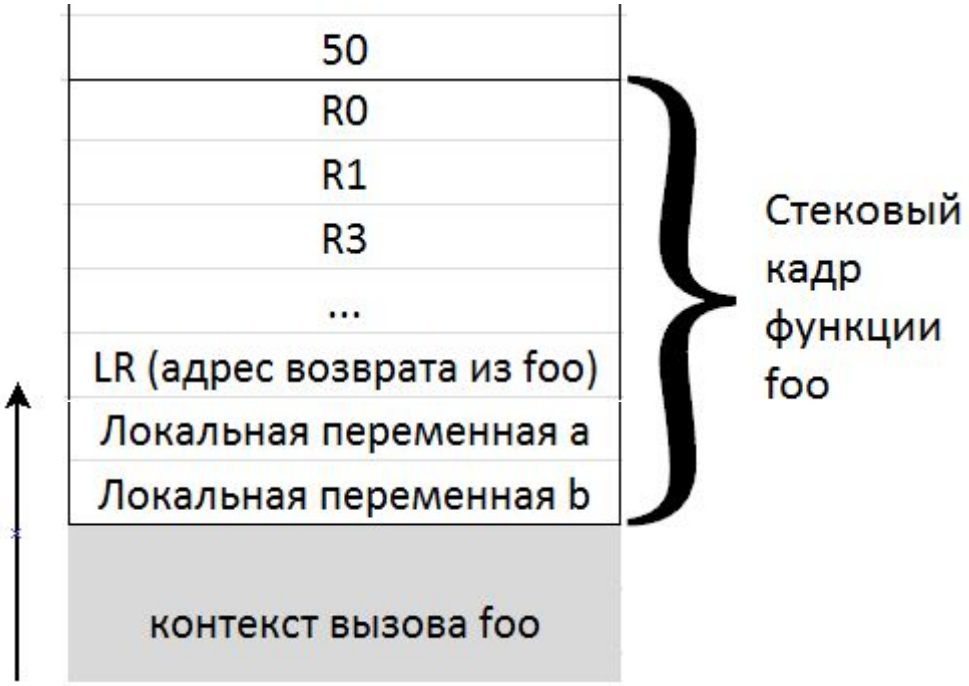
Словно это обычный массив.

Собственно, в ARM стек и есть массив.

Функции в ассемблере: контекст

Упрощенный вид:

```
int foo(int a, int b)
{
1:  b--;
2:  bar(50);
    2.1:  push r0-lr
    2.2:  push 50
    2.3:  BL bar;
3:  a++;
4:  return a;
}
```



Функции в ассемблере: контекст

Упрощенный вид:

```
int bar(int c)
{
1:  buzz(77);
    1.1:  push r0-lr
    1.2:  push 77
    1.3:  BL buzz;
3:  return 0;
}
```

Функции: соглашение о вызове

- Как передаются параметры?
- Как возвращается результат?
- Кто сохраняет контекст?
- Кто восстанавливает контекст?

Все это называется «соглашение о вызове».

Соглашение о вызове может быть разным в зависимости от процессора, ОС, языка, желания левой пятки (fastcall, stdcall...)

Соглашение о вызове

В ARM – ARM Procedure Call Standard
(call convention) (очень упрощенно):

- До четырех параметров передаются на регистрах (r0-r3); остальные через стек
- Контекст сохраняет тот, кого вызвали
- Восстанавливает контекст тот, кого вызвали
- Возвращаемое значение передается через r0 (r0 и r1 для long long и double)

Т.е. слайды 18-19 были только для примера!

Функции (в ARM): краткий итог

- Параметры и локальные переменные хранятся в регистрах или в стеке
- Доступ к переменным в стеке осуществляется через косвенно-регистровую адресацию (например, через SP)
- Перед вызовом функции нужно сохранить контекст, после вызова – восстановить
- Возвращаемое значение передается через регистр r0 (и r1)

Суперскалярная архитектура (конвейеризация)

Идея:

Каждая инструкция ассемблера для процессора – многостадийный процесс.

Разные стадии часто не связаны.

Если их выполнять параллельно, можно получить прирост производительности!

Простой трехстадийный конвейер ARM

Стадии выполнения одной команды:

Выборка из памяти (Fetch)	Декодирование (Decode)	Исполнение (Execute)

Команды	**Такт 1**	**Такт 2**	**Такт 3**	**Такт 4**	**Такт 5**
MOV	Fetch MOV	Decode MOV	Exec MOV		
ADD		Fetch ADD	Decode ADD	Exec ADD	
STR			Fetch STR	Decode STR	Exec STR

Конвеер

Плюсы:

- Процессор загружен равномерно
- Инструкции выполняются параллельно

Минусы:

- Инструкции могут быть зависимы (конфликты)
- Команды перехода срывают конвеер и

Как борются с минусами конвейера?

- Зависимые инструкции:
  - Команда NOP (no operation)
- Долгие инструкции:
  - Внеочередное исполнение
  - И команда NOP
- Срывы из-за переходов:
  - Размотка циклов
  - Условное выполнение вместо условных переходов
  - Inlining функций вместо перехода
  - Предсказание переходов

И еще много всего

Функции

Плюсы:

- Повторное использование кода
- Краткость кода
- Функции – это интерфейс к чужому коду

Минусы:

- Срыв конвейера

Минусы функций: что же делать?

- Чего НЕ НАДО делать:
  - **оптимизировать раньше времени**
  - использовать глобальные переменные вместо параметров
  - бездумно использовать макросы
  - вообще не использовать функции

- Что следует делать:
  - думать **до** того, как писать код
  - написать и отладить, потом оптимизировать
  - включить оптимизацию в компиляторе
  - аккуратно использовать inline и макросы

Чистые функции

Чистая функция (pure) зависит *только* от своих параметров и не меняет глобальное состояние.

Ее результат постоянен.

Например: синус, косинус и т.д.

Чистые функции это хорошо!

Реентерабельные функции

Реентерабельная (reentrant, «повторно входимая») функция *не ломается*, если ее одновременно вызывают несколько потоков.

Чистые функции реентерабельны.