

# Лекция 4

## ARM (продолжение)

# API vs ABI

- API (application programming interface) — спецификация функций, классов, типов данных, протоколов и т. п.
- API определяет функциональность независимо от ее реализации
- API определяется на уровне языка высокого уровня (Си, Си++ и т. п.)
- Примеры: POSIX API (определен в терминах Си), WinAPI

# ABI

- ABI (application binary interface) — интерфейс между ядром ОС, библиотеками на уровне машинного кода
- Нужен для интероперабельности компиляторов (результат компиляции разными компиляторами можно было бы объединить в один исполняемый файл)

# Компоненты ABI

- Размеры базовых типов и правила выравнивания
- Соглашения о вызовах (calling convention)
- Соглашения об именовании объектов
- Соглашения об обработке исключений

# Соглашения о вызовах

- Определяет, как одна подпрограмма вызывает другую подпрограмму
  - Как передаются параметры в подпрограмму
  - Какие регистры общего назначения каким образом можно использовать
  - Как возвращается результат
  - Как используется стек

# Пример: WinAPI stdcall

- Для передачи параметров используется стек
- Аргументы заносятся в стек в обратном порядке
- Результат ( $\leq 32$  бита) возвращается в `eax`
- Стек очищается вызываемой функцией
- На x86 существует около 10 различных соглашений о вызовах

# ARM EABI

- Первые параметры передаются в регистрах r0 ... r3, последующие в стеке
- Параметры размера меньшего int передаются как int
- Double и long long выравниваются по 8 байтам
- Стек должен быть выровнен по 8 байтам
- Если используется стек, он очищается вызывающей подпрограммой
- Регистры r4 ... r11 не должны «портиться» в результате вызова подпрограммы
- Результат возвращается в r0 ... r3

# Вызов подпрограммы

- Инструкция bl (branch with link) или blx  
bl TARGET
- Точка возврата (адрес, по которому размещается инструкция bl + 4) копируется в lr (r14), старое значение lr теряется
- В pc загружается адрес TARGET



# Возврат из подпрограммы

- Несколько вариантов возврата
- Копируем lr в pc  
`mov pc, lr`
- Восстанавливаем pc из стека  
`ldmfd sp!, { r4, pc }`
- Еще...

# Subroutine prologue/epilogue

- Простейший пролог:  
`stmfd sp!, { r4, lr }`
- Простейший эпилог:  
`ldmfd sp!, { r4, pc }`
- Инструкции `ldm[f|e][a|d] reg[!]`, REGS сохраняют множество регистров
  - ! означает, что значение регистра должно обновиться после выполнения операции
  - F — full, E — empty, a — ascending, d — descending определяют разные варианты стека
- Регистры сохраняются так, что регистр с меньшим номером находится по меньшим адресам

# Работа с памятью

- Инструкции ldr и str загружают 32-битное значение из памяти в регистр и сохраняют значение в память  
ldr r0, [r0]
- Возможные варианты задания адреса в памяти определяются поддерживаемыми режимами адресации процессора

# Режимы адресации

- Pre-indexed
  - $[breg]$  — адрес находится в  $breg$
  - $[breg, offset]$  — адрес равен  $breg + offset$
  - $[breg, ireg]$  — адрес равен  $breg + ireg$
  - $[breg, -ireg]$  — адрес равен  $breg - ireg$
  - $[breg, ireg, shift\ cnt]$  —  $breg + ireg\ shift\ cnt$   
варианты сдвига рассмотрим далее
  - $[breg, -ireg, shift\ cnt]$

# Режимы адресации

- Pre-indexed writeback — вычисленный адрес записывается в регистр базы
  - [breg]!
  - [breg, offset]!
  - [breg, [-]ireg]!
  - [breg, [-]ireg, shift cnt]!

# Режимы адресации

- Post-indexed — используется базовый регистр, но после обращения к памяти его значение модифицируется
  - [breg], offset
  - [breg], [-]ireg
  - [breg], [-]ireg, shift cnt

# Режимы адресации

- Примеры

Ldr r0, [r0] // загрузить r0 значением по  
// старому адресу r0

Stm r1, [sp, #-4]! // сохранение r1 в стек

Ldm r1, [sp], #4 // восстановление r1 из стека

Ldm r2, [r3, r4, lsl #2] // адрес в памяти:

//  $r3 + (r4 \ll 2)$  — работа с массивом

# Загрузка/сохранение байтов

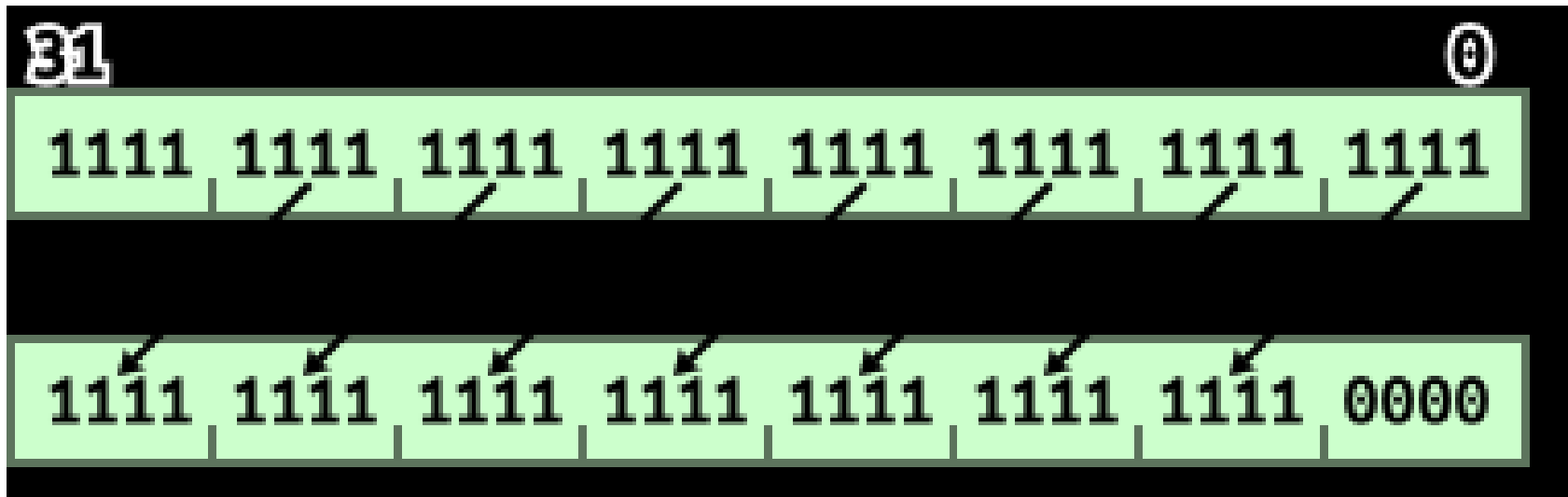
Ldrb	r0, [r1]	// загрузить беззнаковый байт
Ldrsb	r0, [r1]	// загрузить знаковый байт
Strb	r0, [r1]	// сохранить байт
Ldrh	r0, [r1]	// загрузить unsigned short
Ldrsh	r0, [r1]	// загрузить signed short
Strh	r0, [r1]	// сохранить short



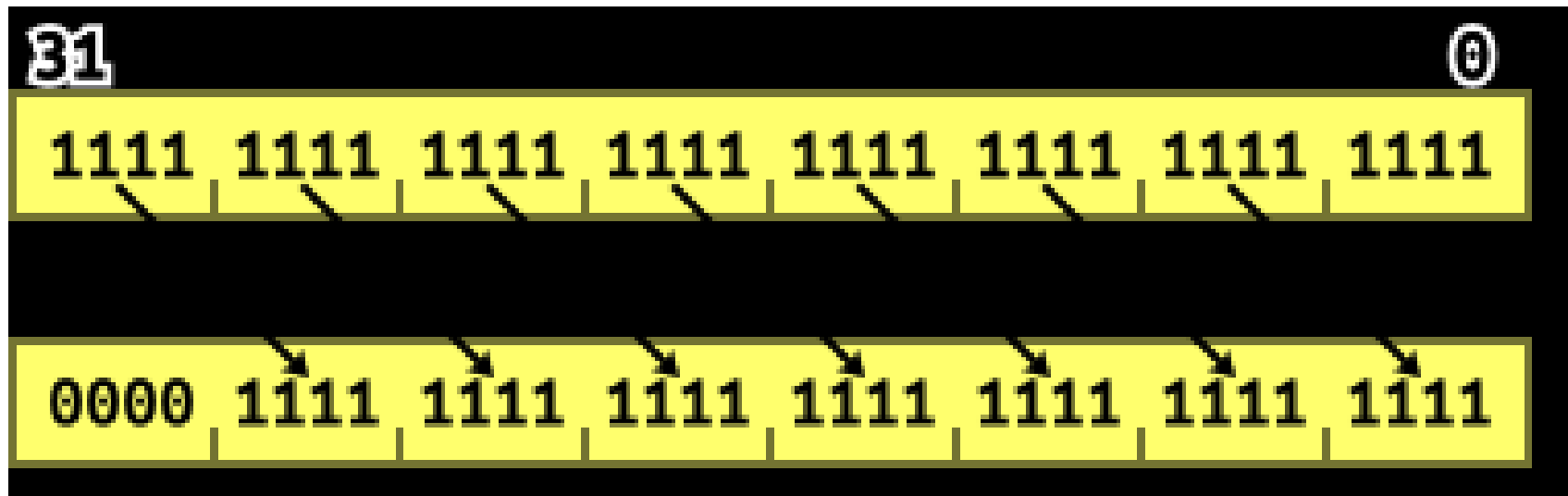
# ВОЗМОЖНЫЕ СДВИГИ

- Lsl (logical shift left)
- Lsr (logical shift right)
- Asr (arithmetical shift right)
- Ror (rotate right) — циклический сдвиг вправо
- Rrx (rotate right extended) — циклический сдвиг через флаг C — сдвиг только на 1 бит!

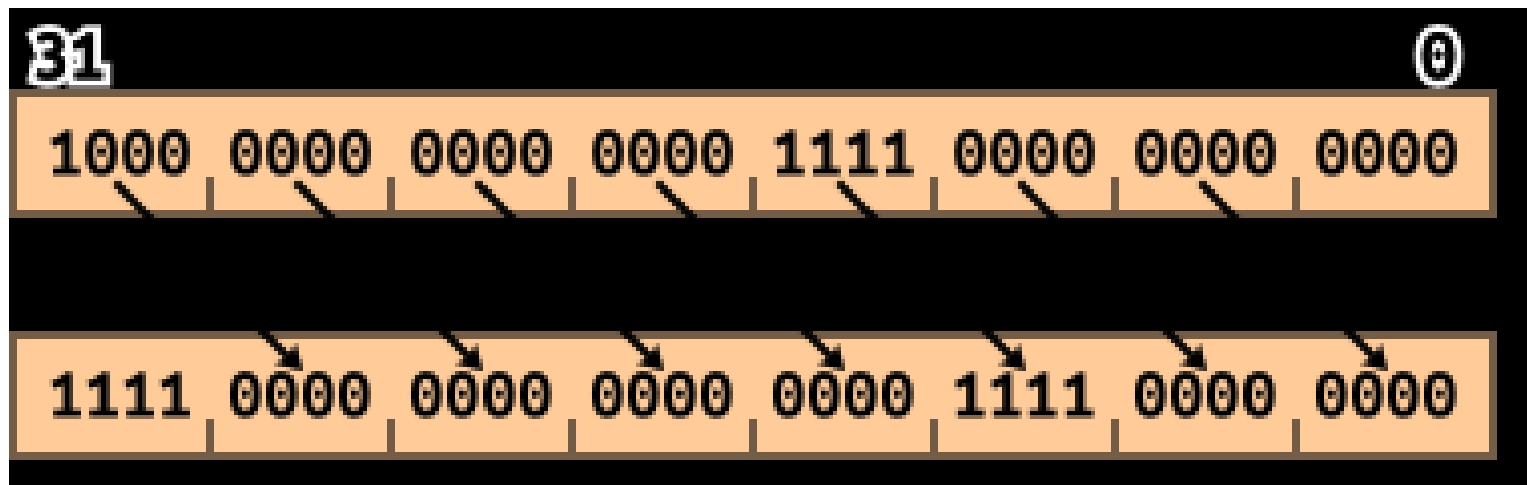
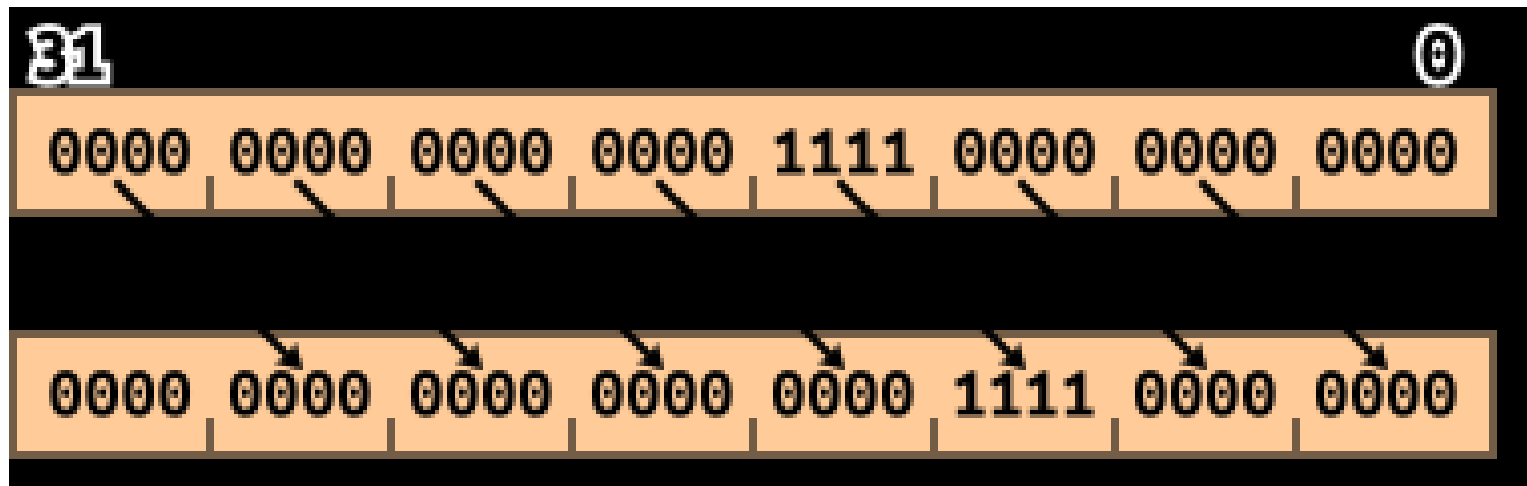
# Logical Shift Left



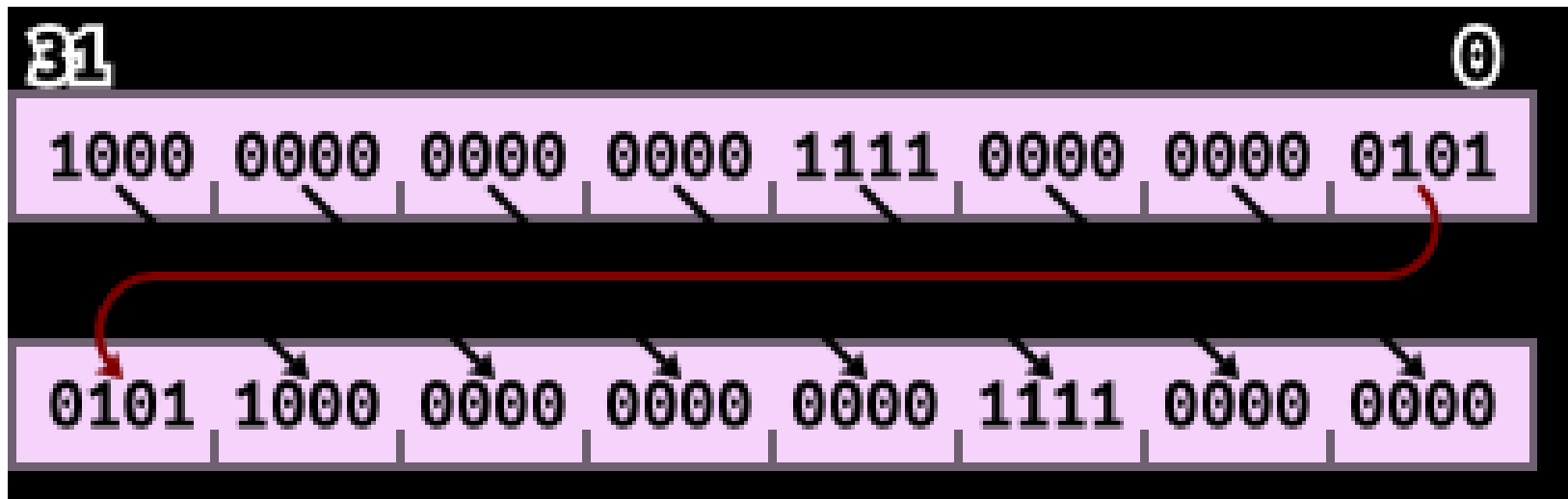
# Logical Shift Right



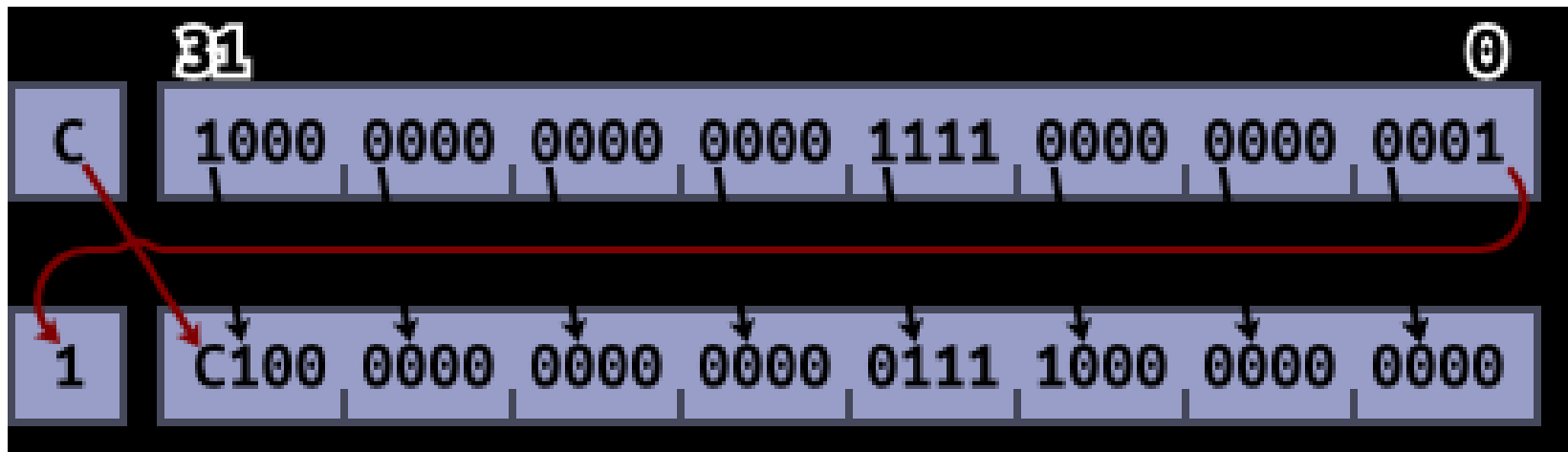
# Arithmetical Shift Right



# Rotate Right



# Rotate Right Extended



# Barrel Shifter

- Операции сдвига могут использовать вместе со вторым аргументом бинарных операций или с пересылкой
- С пересылкой:  
`mov r1, r2, lsl #1` //  $r1 = r2 \ll 1$
- С бинарной операцией:  
`add r3, r2, r1, lsr #2` //  $r3 = r2 + r1 \gg 2$

# CPSR (current program status register)

- Содержит флаги, отвечающие за разные режимы работы процессора (запрет прерываний, уровни привилегий)
- Содержит арифметические флаги 'Z', 'N', 'C', 'V'
- Флаг 'Z' (zero) — устанавливается в 1, если результат операции равен 0
- Флаг 'N' (negative) — устанавливается в 1, если результат операции отрицателен (то есть старший бит равен 1)



# Флаги C, V

- Флаг C устанавливается, если при выполнении операции произошло переполнение разрядной сетки беззнаковых чисел
- Флаг V устанавливается при переполнении знаковых чисел

# Пример

- Рассмотрим операции с числами размера 8:

$$120 + 54 = 174 \quad (-82) : Z = 0, N = 1, C = 0, V = 1$$

$$140 \quad (-116) + 130 \quad (-126) = 14 : Z=0, N=0, C=1, \\ V=1$$

$$20 + 250 \quad (-6) = 14 : Z=0, N=0, C=1, V=0$$

# Установка флагов

- Инструкции сравнения (cmp, tst, ...) устанавливают флаги в зависимости от результата операции
- Обычные арифметические и логические операции не устанавливают флаги
- Чтобы устанавливали — суффикс 's'

Add     r0, r1, r2     // флаги не устанавливаются

Adds    r0, r1, r2     // флаги устанавливаются

# Условное выполнение

- Каждая инструкция может быть пропущена в зависимости от условия

Add     r1, r2, r3     // выполняется всегда

Addeq r1, r2, r3     // выполняется, если 'z'

- Инструкция перехода аналогична

B        label        // безусловный переход

Bne     label        // переход, если  $z \neq 0$

# Поддерживаемые условия

- EQ — равно (нуль)
- NE — не равно (не нуль)
- HS или CS — беззнаковое  $\geq$
- LO или CC — беззнаковое  $<$
- MI — знаковое  $< 0$
- PL — знаковое  $\geq 0$
- VS — переполнение
- VC — нет переполнения

# Поддерживаемые условия

- HI — беззнаковое  $>$
- LS — беззнаковое  $\leq$
- GE — знаковое  $\geq$
- LT — знаковое  $<$
- GT — знаковое  $>$
- LE — знаковое  $\leq$