АРХИТЕКТУРА СОВРЕМЕННЫХ ЭВМ

Лекция 12:
Архитектура процессора ARM для встроенных систем

ВМиК МГУ им. М.В. Ломоносова, Кафедра АСВК
Чл.-корр., профессор, д.ф.-м.н. Королёв Л.Н.,
Ассистент Волканов Д.Ю.

План

- Введение в ARM

  Парадигма программирования

  Набор инструкций

  Архитектура системы

ARM Powered Products

План

Введение в ARM

- Парадигма программирования

Набор инструкций

Архитектура системы

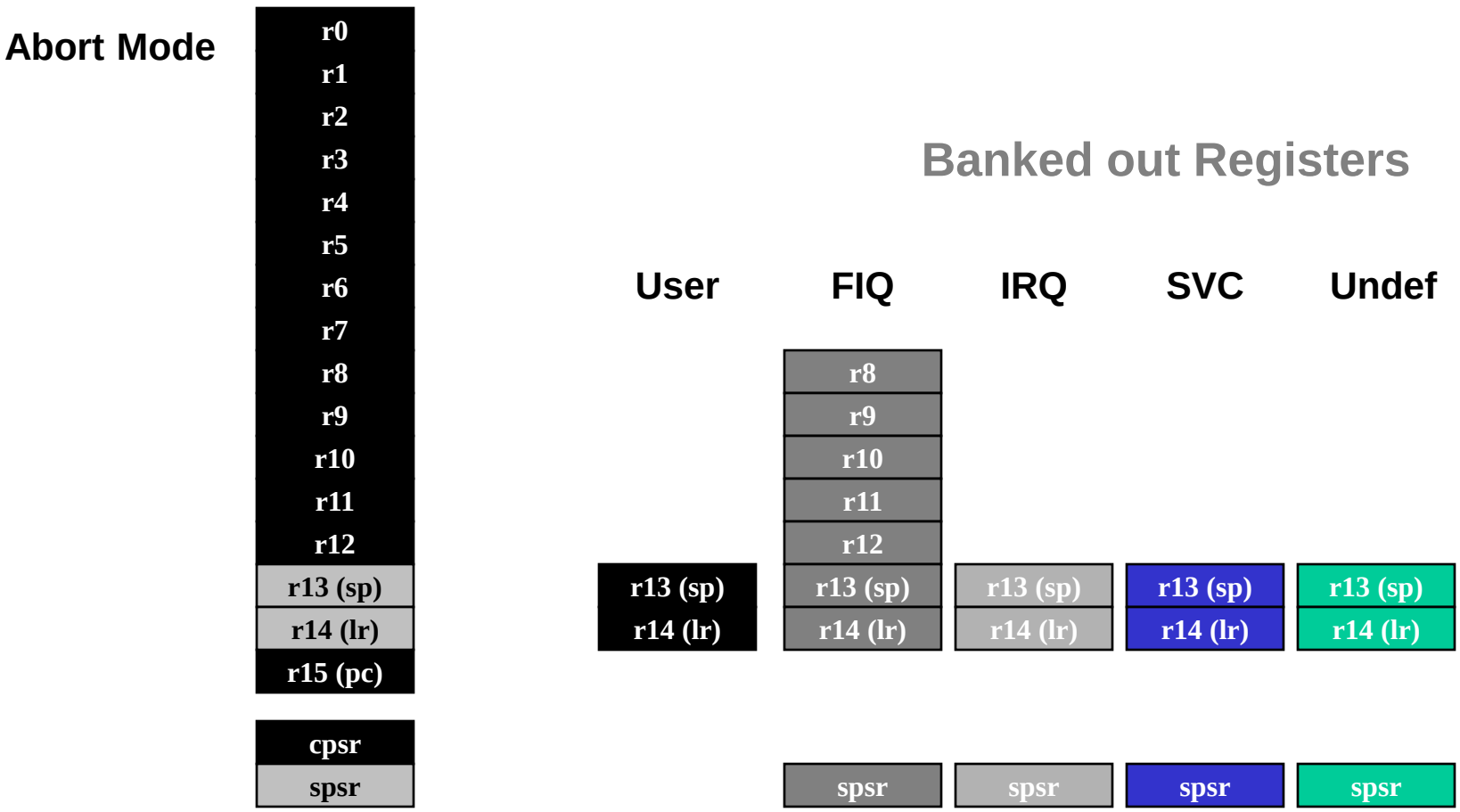
Размер типов данных и набор иструкций

- ARM имеет 32-битную архитектуру.
- Обычно в ARM используется следующие ключевые слова:
  - **Byte** - 8 bits
  - **Halfword** - 16 bits (два байта)
  - **Word** - 32 bits (четыре байта)
- Большинство ARM процессоров реализует два набора инструкций
  - 32-bit ARM Instruction Set
  - 16-bit Thumb Instruction Set

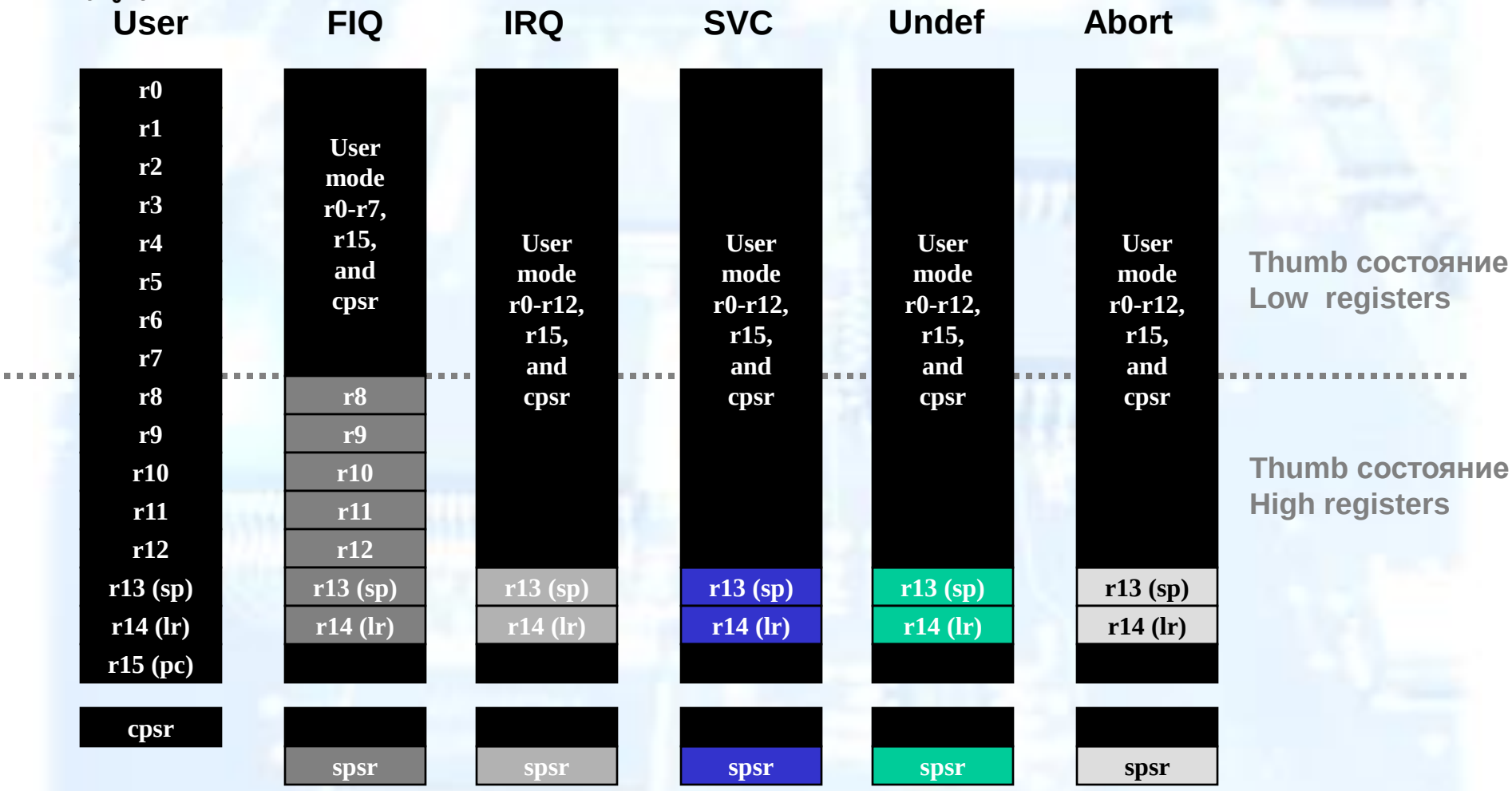
Режимы работы процессора

- Семь основных режимов функционирования ARM:
  - **User** : непривилегированный режим, под которым выполняется большинство задач
  - **FIQ** : включается, когда приходит high priority (fast) прерывание
  - **IRQ** : включается, когда приходит low priority (normal) прерывание
  - **Supervisor** : включается при перегрузке и когда выполняется Software Interrupt instruction
  - **Abort** : позволяется ловить нарушения режима доступа к памяти
  - **Undef** : позволяет ловить нераспознанные инструкции
  - **System** : привилегированный режим использующий те же регистры, что и User режим

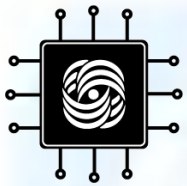
Набор регистров в ARM



Организация регистров



Note: System mode использует те же регистры, что и User mode



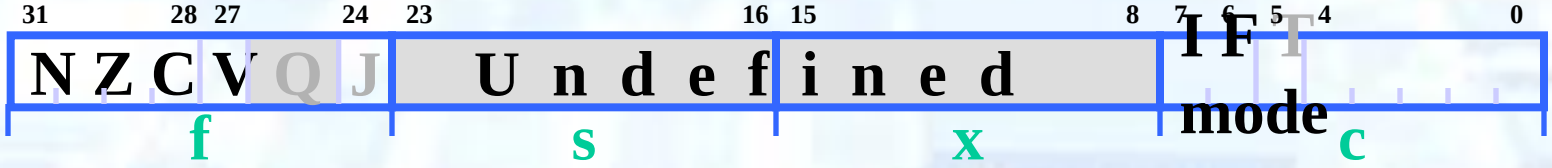
Типы регистров

- В ARM есть 37 регистров размером 32-bits.
  - 1 специальный регистр: program counter
  - 1 специальный регистр: current program status
  - 5 специальных регистров для хранения program status
  - 30 регистров общего назначения

- В любом режиме работы процессора имеется доступ к следующим регистрам:
  - r0-r12 РОН
  - r13 (the stack pointer, sp) иr14 (the link register, lr)
  - program counter, r15 (pc)
  - current program status register, cpsr

  Привилегированный (except System) режим может обращаться к spsr (saved program status register)

Регистры состояния программы



- Флаги условных переходов
  - N = **N**egative вычисляется АЛУ
  - Z = **Z**ero вычисляется АЛУ
  - C = АЛУ операция **C**arried out
  - V = АЛУ операция o**V**erflowed
- Sticky Overflow флаг - Q flag
  - Только для 5TE/J архитертур
  - Определяет насыщение
- J bit
  - Только для 5TEJ архитектур
  - J = 1: процессор в состоянии Jazelle
- Биты отключения прерываний.
  - I = 1: отключает IRQ.
  - F = 1: отключает FIQ.
- T Bit
  - Только для xT аржитетур
  - T = 0: процессор в состоянии ARM
  - T = 1: процессор в состоянии Thumb
- Mode bits
  - Указывают режим работы процессора

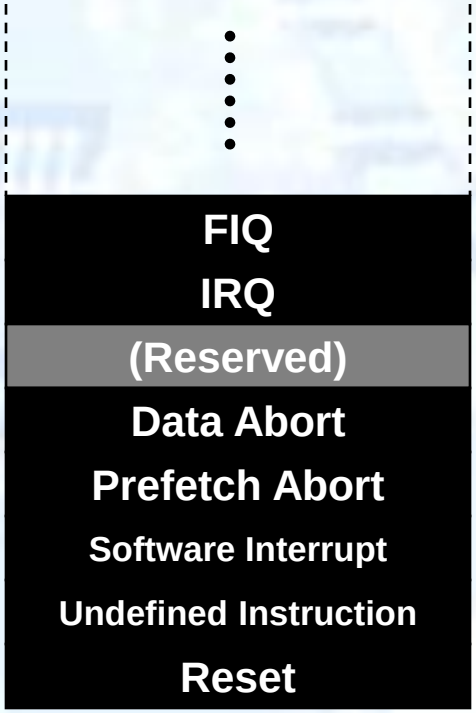
Program Counter (r15)

- Если процессор находится в режиме ARM:
  - Все инструкции размером 32 бита
  - Все инструкции должны быть выровнены по слову (word aligned)
- Если процессор находится в режиме Thumb:
  - Все инструкции размером 16 бит
  - Все инструкции должны быть выровнены по полуслову (halfword aligned)
- Если процессор находится в режиме Jazelle:
  - Все инструкции размером 8 бит
  - Процессор позволяется читать по 4 инструкции сразу

Обработка исключений

- Алгоритм обработки исключений:
  - Копируется CPSR в SPSR_<mode>
  - Заполняются CPSR биты
    - Состояние изменяется на ARM
    - Включается exception mode
    - Игнорируются прерывания (if appropriate)
  - Stores the return address in LR_<mode>
  - Sets PC to vector address
- Для возврата к нормальной работе:
  - Восстанавливается CPSR из SPSR_<mode>
  - Восстанавливается PC из LR_<mode>

Все это может проделываться только в состоянии ARM.



Vector Table

Vector table может находится по адресу 0xFFFF0000 на ARM720T и ARM9/10 семействе устройств

Разработка ARM архитектуры

1

2

3

Ранние ARM архитектуры

4

Поддержка Halfword

signed halfword / байтов

System mode

SA-110

SA-1110

4T

Thumb instruction set

ARM7TDMI **ARM9TDMI**

ARM720T **ARM940T**

5TE

Improved ARM/Thumb Interworking

CLZ

Saturated maths

DSP multiply-accumulate instructions

ARM1020E

XScale

ARM9E-S

ARM966E-S

5TEJ

Jazelle

выполнение Java bytecode

ARM9EJ-S **ARM926EJ-S**

ARM7EJ-S **ARM1026EJ-S**

6

SIMD Instructions

Multi-processing

V6 Memory architecture (VMSA)

Unaligned data support

ARM1136EJ-S

План

Введение в ARM

Парадигма программирования

- Набор инструкций

Архитектура системы

Условные переходы и флаги

- ARM инструкции могут выполнятся условно путем проставления постфикса с кодом условия.

```
–   CMP   r3,#0
      BEQ   skip
      ADD   r0,r1,r2
    skip
```

```
CMP   r3,#0
  ADDNE r0,r1,r2
```

- По умолчанию, инструкции обработки данных не влияют на условные флаги, но данные флаги могут быть опционально установлены используя “S”. CMP не нуждается в“S”.

```
loop
  …
  SUBS r1,r1,#1
  BNE loop
```

декрементируем r1 и устанавливаем флаги

если Z флаг нулевой, то осуществляем переход

Условные коды

- Возможные условные коды приведены ниже:

Суффикс	Описание	Флаг
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

Примеры условного выполнения

- Использование последовательности условных инструкций

```
if (a==0) func(1);
    CMP     r0,#0
    MOVEQ   r0,#1
    BLEQ    func
```

- Установка флагов, после использование различных условных кодов

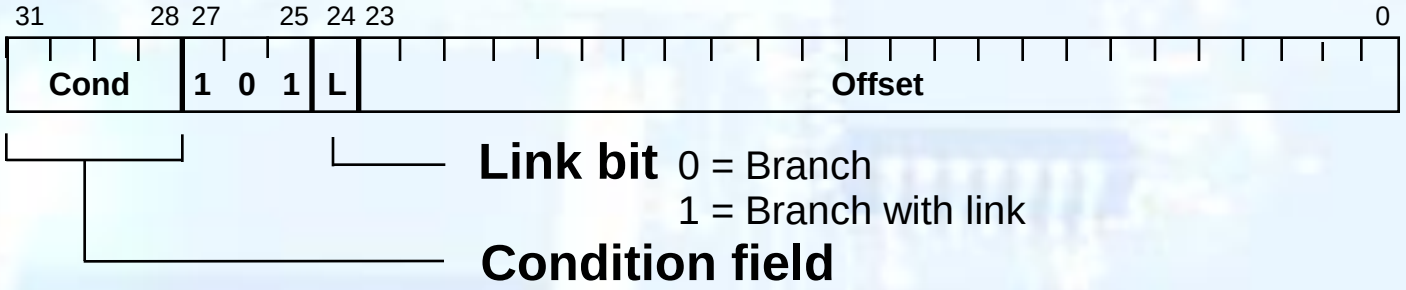
```
if (a==0) x=0;
if (a>0)  x=1;
    CMP     r0,#0
    MOVEQ   r1,#0
    MOVGT   r1,#1
```

- Использование условных инструкций сравнения

```
if (a==4 || a==10) x=0;
    CMP     r0,#4
    CMPNE   r0,#10
    MOVEQ   r1,#0
```

Инструкции ветвления

- Branch : `B{<cond>} label`
- Branch со связью: `BL{<cond>} subroutine_label`



Инструкции обработки данных

- Состоят из:
  - Арифметических: `ADD` `ADC` `SUB` `SBC` `RSB` `RSC`
  - Логических: `AND` `ORR` `EOR` `BIC`
  - Сравнений: `CMP` `CMN` `TST` `TEQ`
  - Перемещения данных: `MOV` `MVN`

- Данные инструкции работают только с регистрами, НЕ с памятью.

- Синтаксис:

```
<Operation>{<cond>}{S} Rd, Rn, Operand2
```

  - Сравнения только устанавливают флаги
  - Перемещение данных не специфицирует Rn

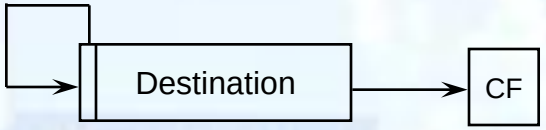
- Второй операнд отправляется на АЛУ через barrel shifter.

Barrel Shifter

LSL : Logical Left Shift

Умножение на 2

ASR: Arithmetic Right Shift

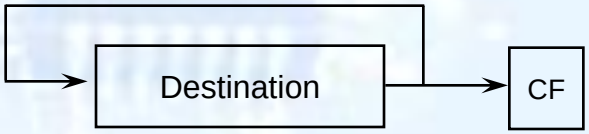


Деление на 2,
сохраняя бит флага

LSR : Logical Shift Right

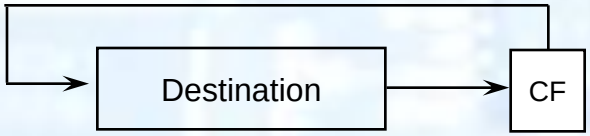
Деление на 2

ROR: Rotate Right



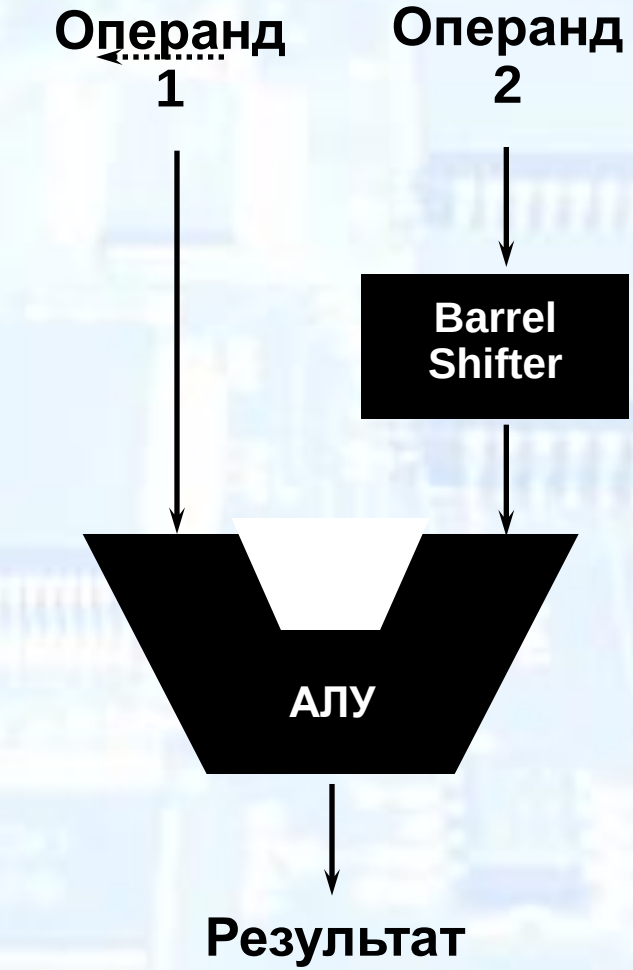
Циклическое смещение бита от LSB к MSB

RRX: Rotate Right Extended



Циклическое смещение через CF к MSB

Использование Barrel Shifter: Второй операнд



Умножение

- Синтаксис:
  - MUL{<cond>}{S} Rd, Rm, Rs — Rd = Rm * Rs
  - MLA{<cond>}{S} Rd,Rm,Rs,Rn — Rd = (Rm * Rs) + Rn
  - [U|S]MULL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := Rm*Rs
  - [U|S]MLAL{<cond>}{S} RdLo, RdHi, Rm, Rs — RdHi,RdLo := (Rm*Rs)+RdHi,RdLo
- Время в циклах
  - Основная MUL инструкция
    - 2-5 циклов на ARM7TDMI
    - 1-3 циклов на StrongARM/XScale
    - 2 цикла на ARM9E/ARM102xE
  - +1 цикл для ARM9TDMI (over ARM7TDMI)
  - +1 цикл для “long”

Помещение данных в регистр

LDR STR	Word	
LDRB	STRB	Byte
LDRH	STRH	Halfword
LDRSB		Signed byte load
LDRSH		Signed halfword load

- Память должна поддерживать все допустимые размеры
- Синтаксис:
  - LDR{<cond>}{<size>} Rd, <address>
  - STR{<cond>}{<size>} Rd, <address>

e.g. LDREQB

Доступ по адресу

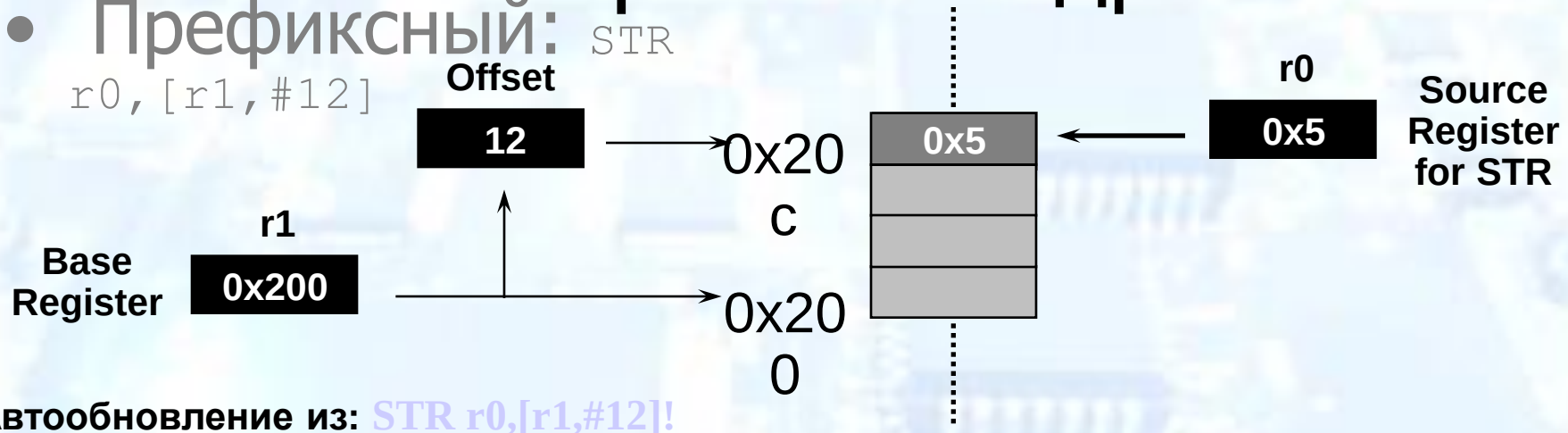
- Адрес доступные по LDR/STR определяется как значение регистра плюс смещение
- Для слова и беззнакового байта доступа, смещение может быть
  - 0 - 4095 bytes

```
LDR r0,[r1,#8]
```

- Для полуслова и знакового полуслова, смещение может быть :
  - 0-255 bytes.
  - регистр

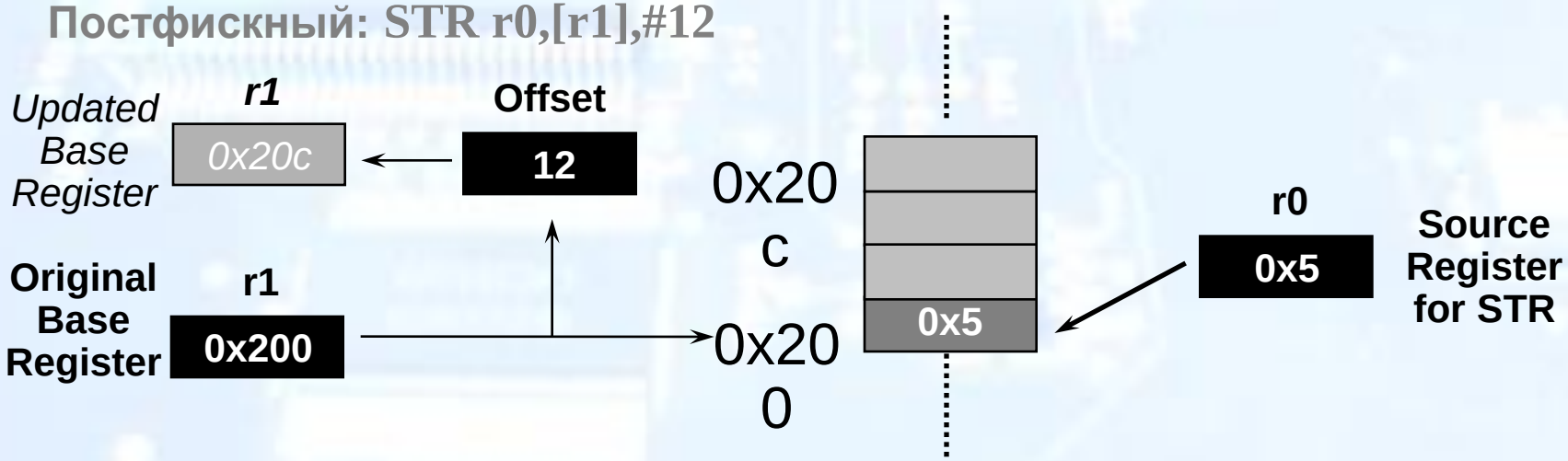
Префиксный или постфиксный адрес?

- Префиксный: `STR r0,[r1,#12]`



Автообновление из: `STR r0,[r1,#12]!`

- **Постфискный: STR r0,[r1],#12**



LDM / STM

- Синтаксис:

  `<LDM|STM>`{<cond>}<addressing_mode> Rb{!}, <register list>

- 4 режима адресования:

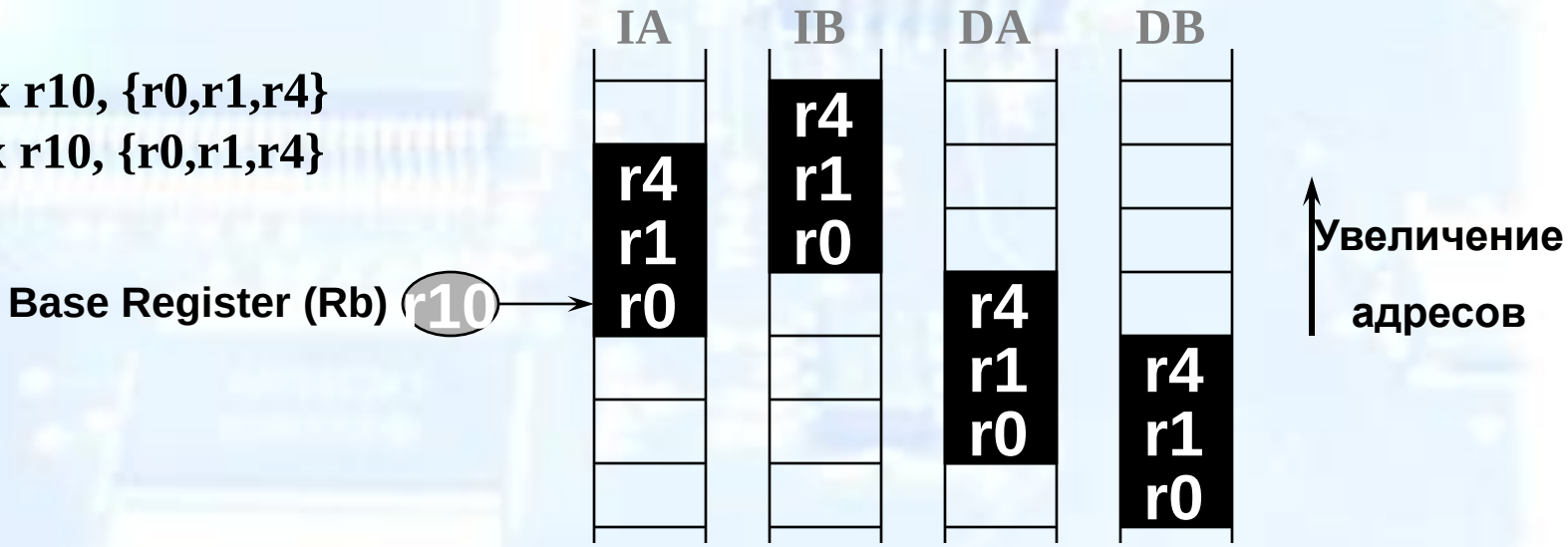
  LDMIA / STMIA инкрементить после

  LDMIB / STMIB инкрементить до

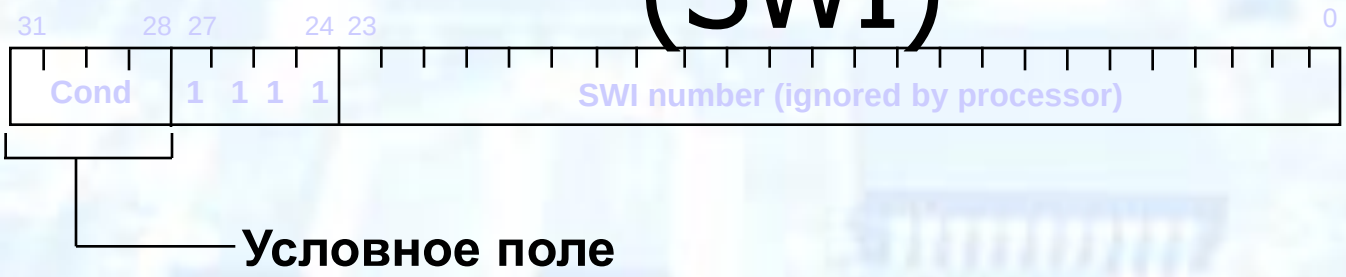
  LDMDA / STMDA декрементить после

  LDMDB / STMDB декрементить до

LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}

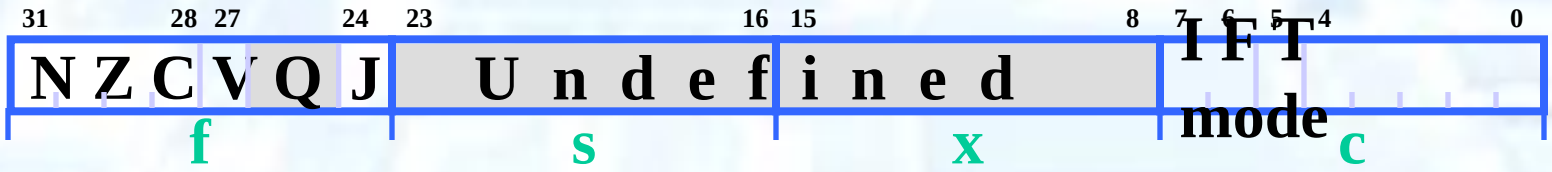


Программное прерывание (SWI)



- Возбуждает обработчик прерываний в соответствии с SWI hardware vector
- SWI handler может определить SWI number чтобы решить какую операцию надо выполнить
- Используя SWI механизм, ОС может реализовать набор привилегированных операций
- Синтаксис:
  - `SWI{<cond>} <SWI number>`

PSR инструкции



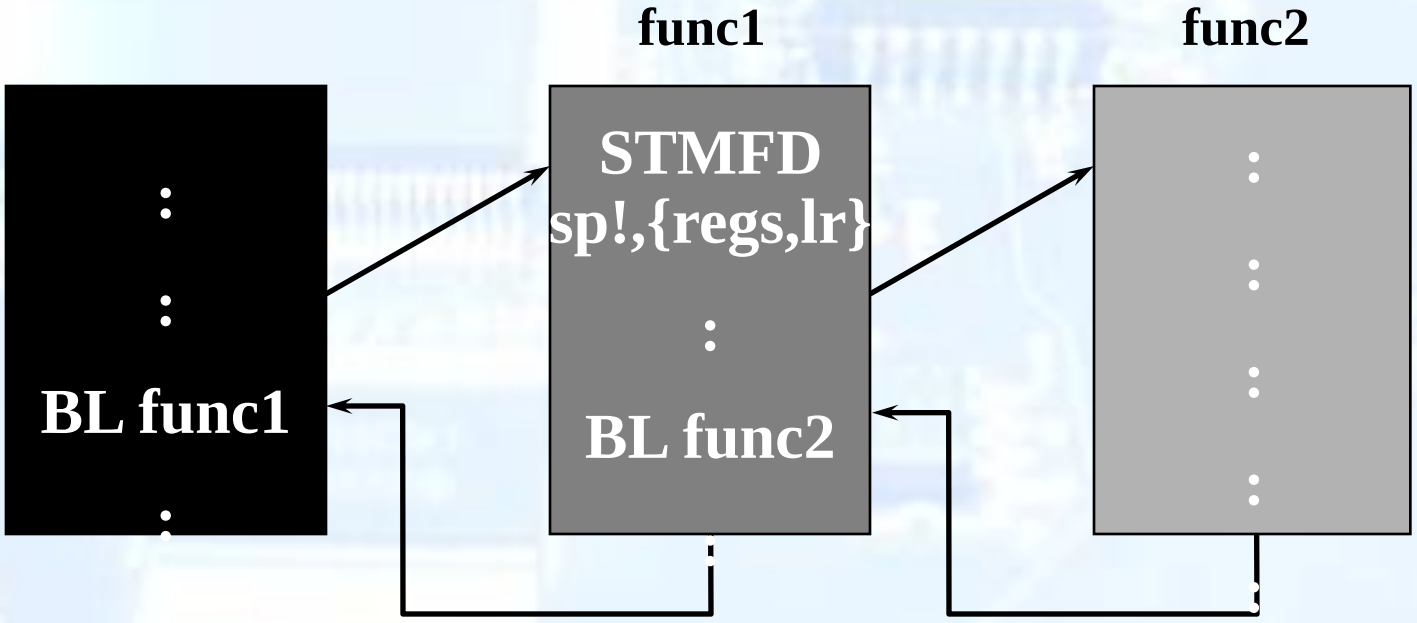
- MRS и MSR позволяет переместить содержимое CPSR / SPSR в или из регистра общего назначения
- Синтаксис:
  - `MRS{<cond>} Rd,<psr>  ; Rd = <psr>`
  - `MSR{<cond>} <psr[_fields]>,Rm  ; <psr[_fields]> = Rm`

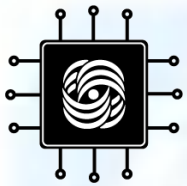
где
  - `<psr> = CPSR or SPSR`
  - `[_fields] = any combination of 'fsxc'`

- Так же возможно
  - `MSR{<cond>} <psr_fields>,#Immediate`

ARM ветви

- B <label>
  - PC relative. ±32 Mbyte range.
- BL <subroutine>
  - Хранит и возвращает адрес в LR





План

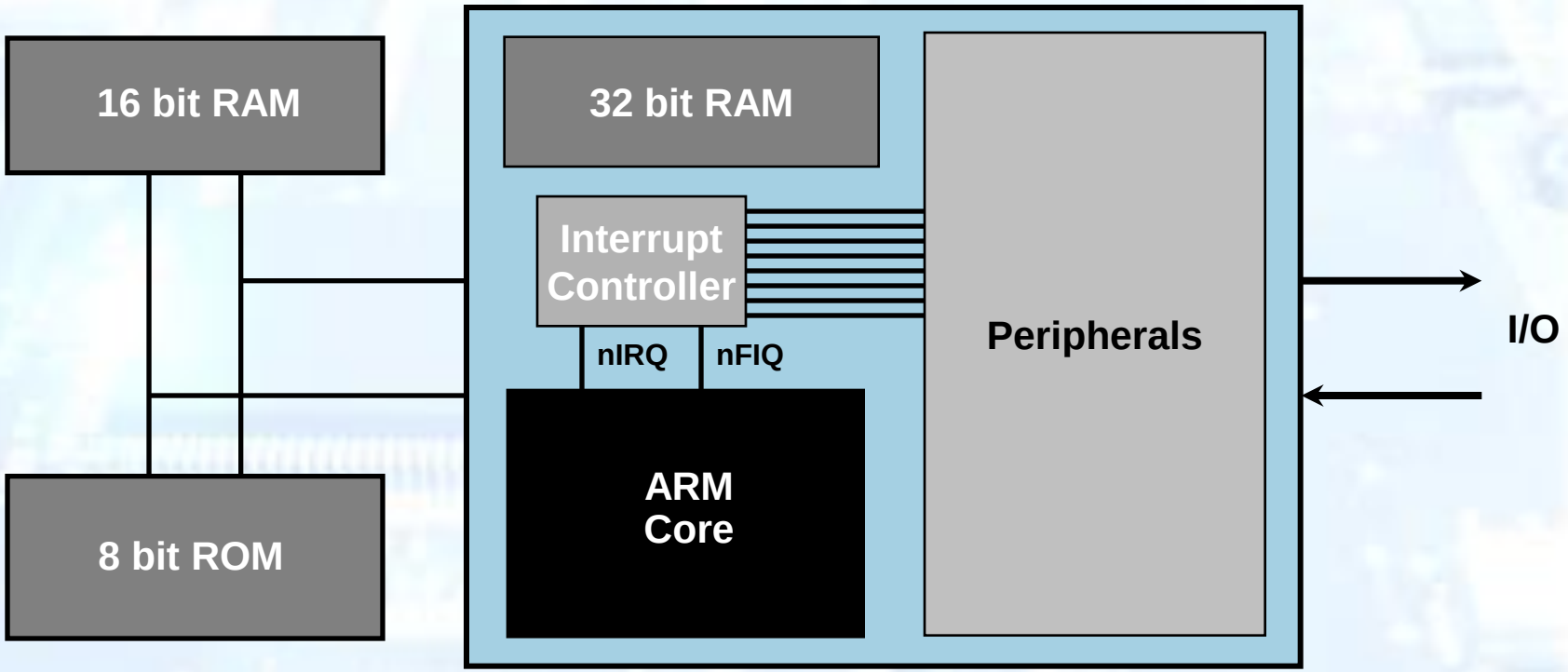
Введение в ARM

Парадигма программирования

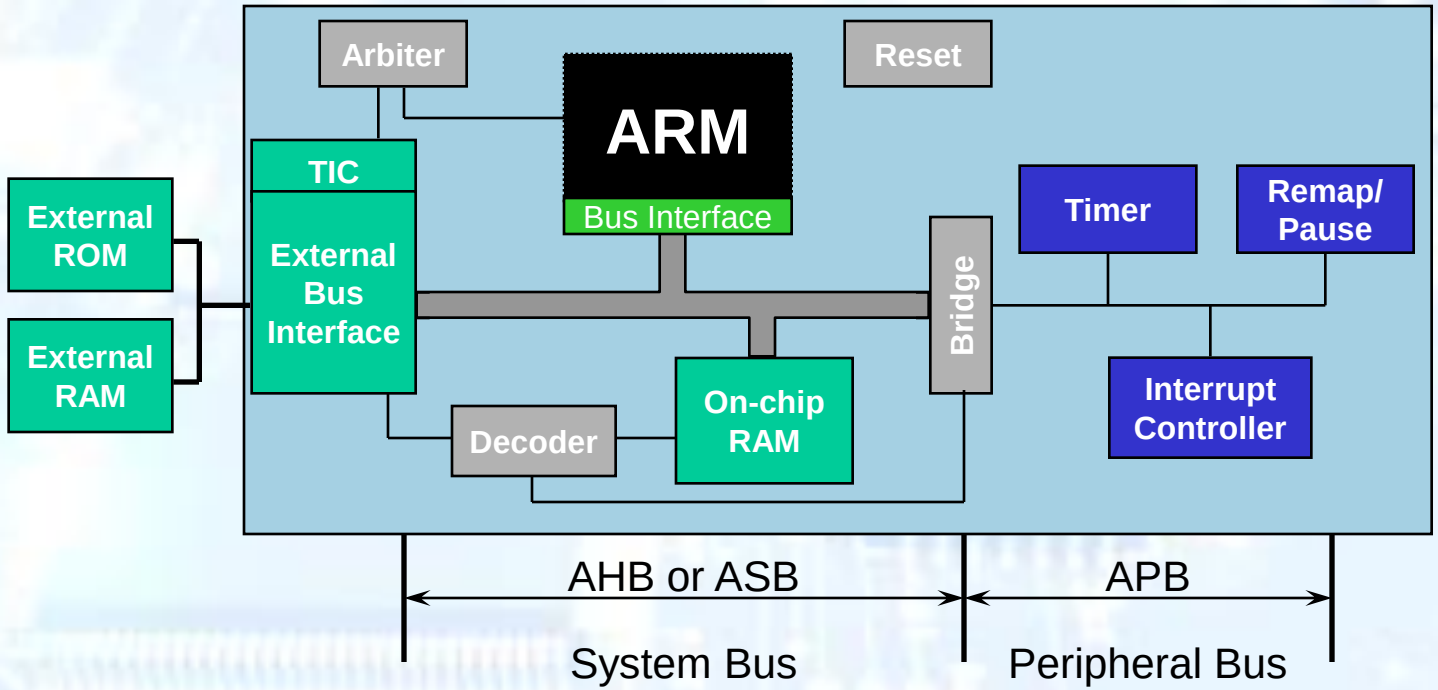
Набор инструкций

- Архитектура системы

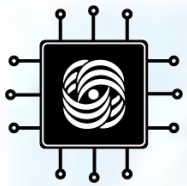
Пример ARM-based системы



AMBA



- AMBA
  - Advanced Microcontroller Bus Architecture
- ADK
  - Complete AMBA Design Kit
- ACT
  - AMBA Compliance Testbench
- PrimeCell
  - ARM's AMBA compliant peripherals



Спасибо за внимание!