ВСТРОЕННЫЕ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Лекция 4:
Архитектура процессоров для ВИУСРВ

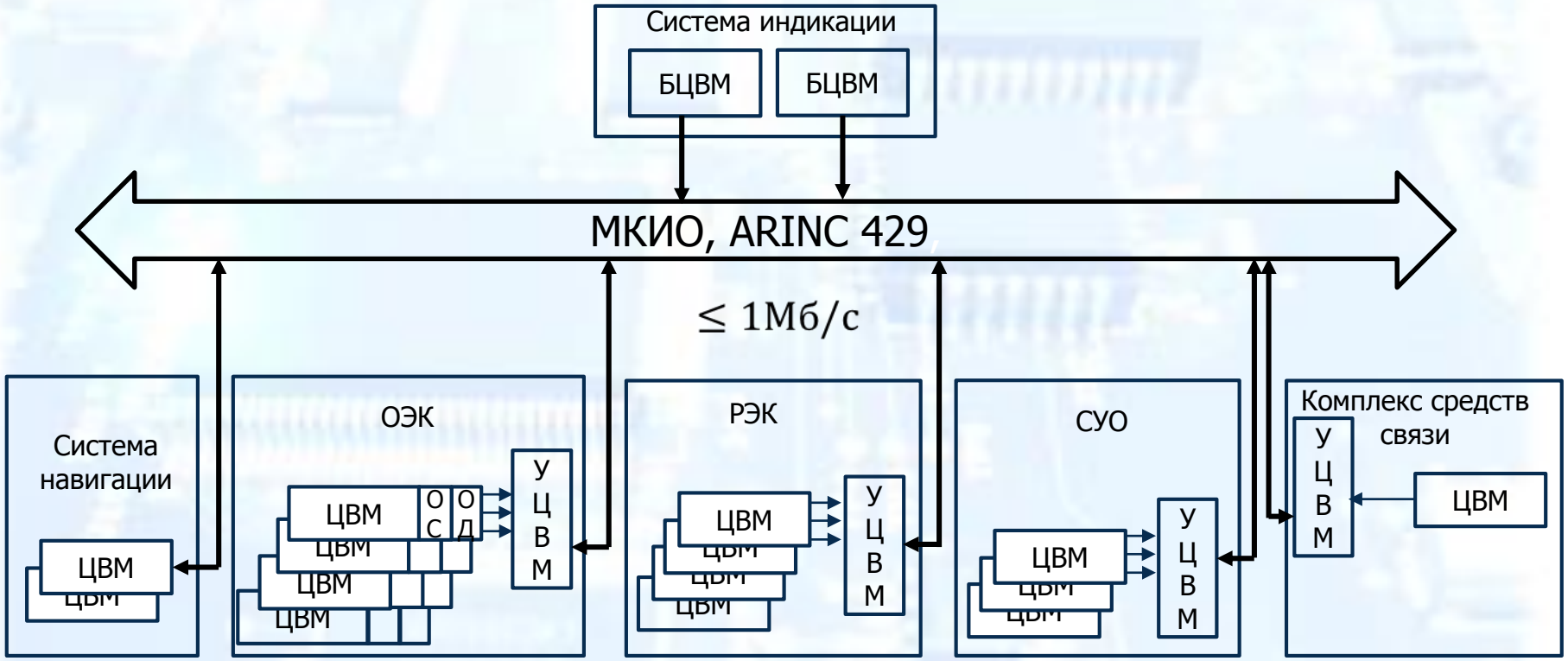
ВМиК МГУ им. М.В. Ломоносова, Кафедра АСВК,
Лаборатория Вычислительных Комплексов,
Ассистент Волканов Д.Ю.

План

- Интегрированная модульная авионика
- Процессор ARM
- Процессор обработки сигналов (NM 6403)
- Задача WCET

4-е поколение боевых комплексов

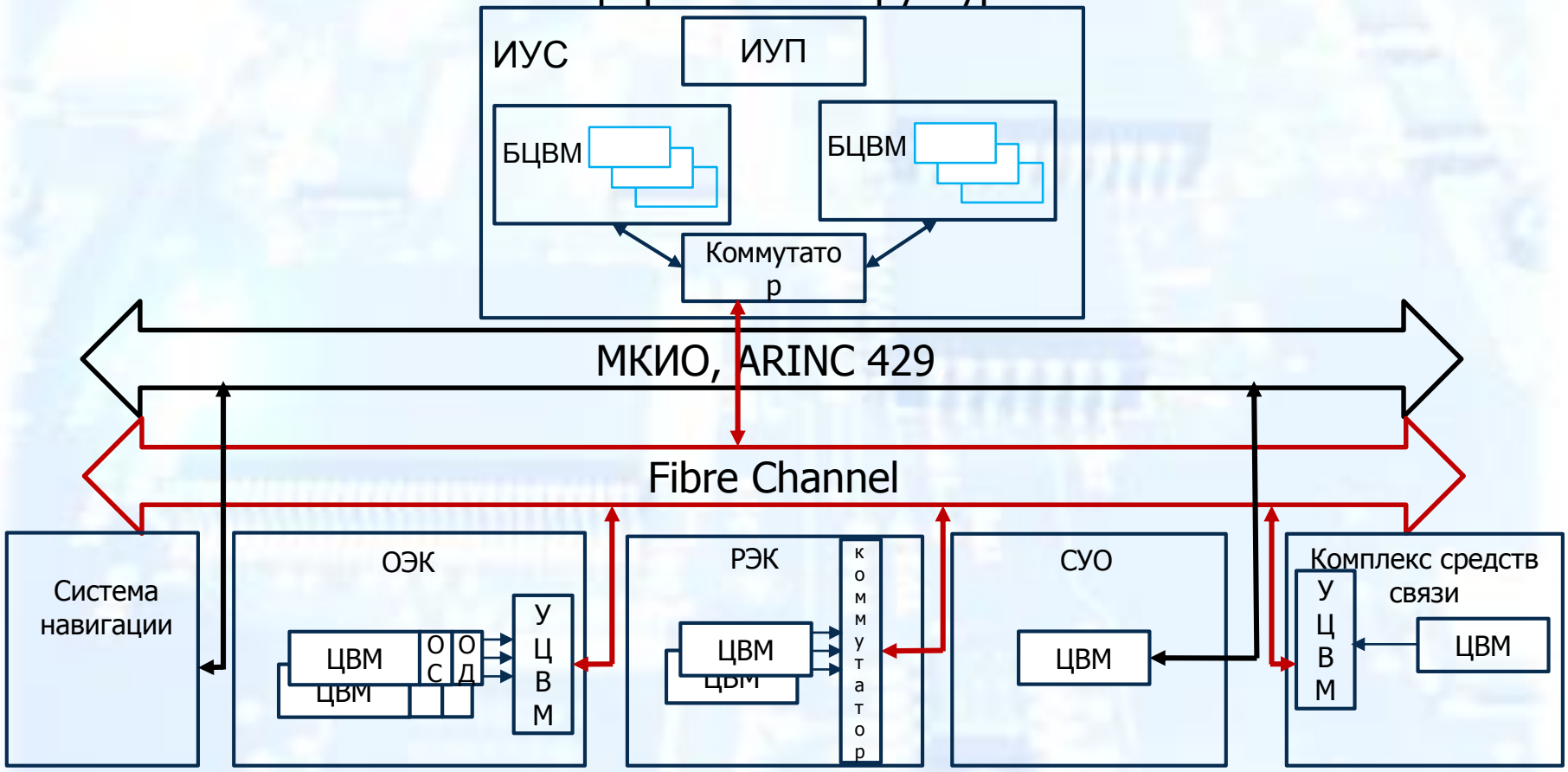
Федеративная структура КБО



- Низкоскоростные каналы информационного обмена (МКИО, ARINC 429) Большое количество разнотипных специализированных вычислителей
- Невозможность комплексной обработки информации
- Применение однопроцессорных вычислительных систем

3

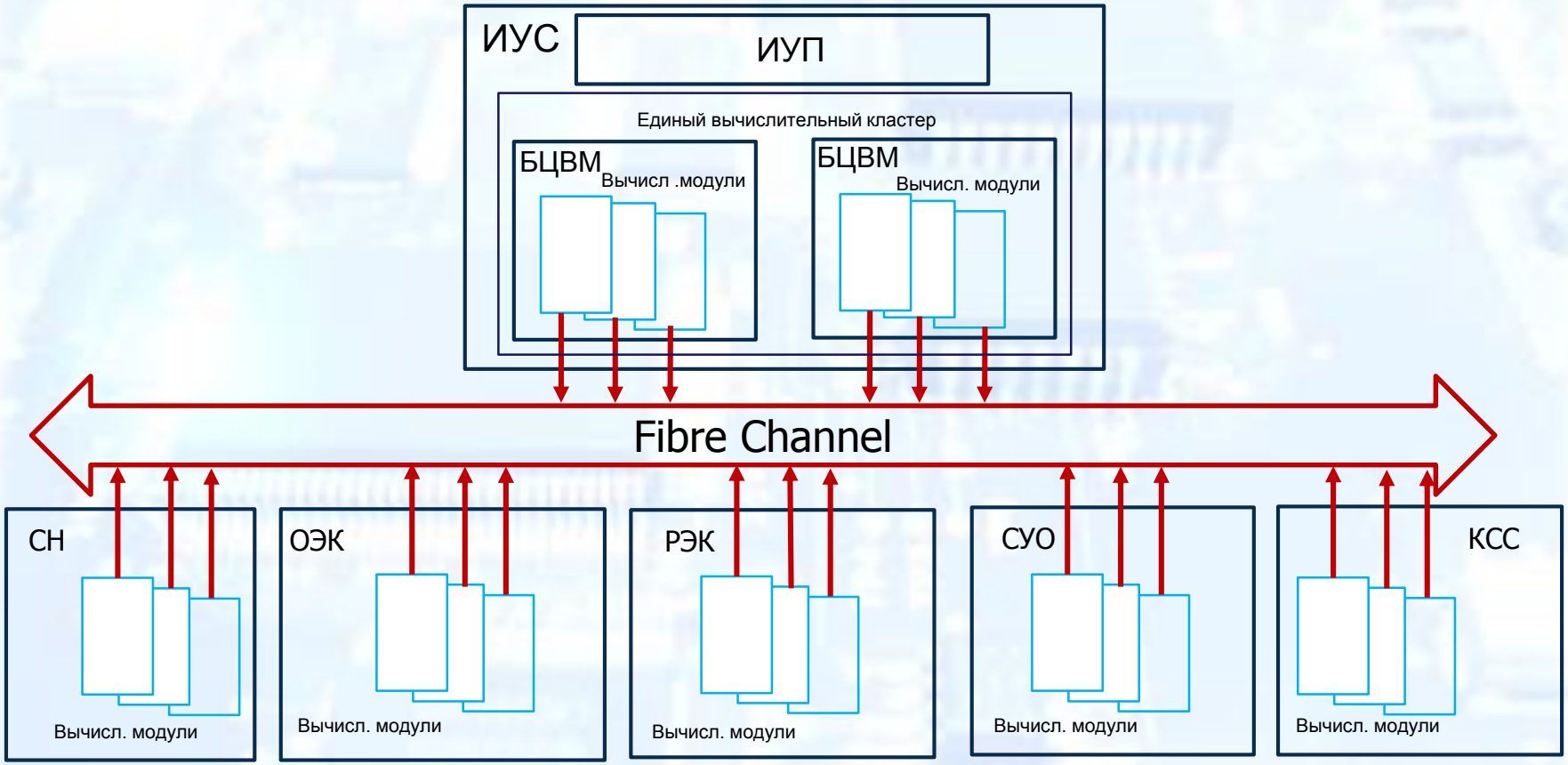
5-е поколение боевых комплексов
Интегрированная структура КБО



- Использование высокоскоростных каналов информационного обмена наряду с МКИО
- Сокращение количества отдельных вычислителей
- Комплексная обработка информации
- Перенос третичной и вторичной (частично) обработки информации в «ядро»
- Применение многопроцессорных вычислительных систем

4

5+ поколение боевых комплексов
Интегрированная модульная авионика боевых комплексов



- Использование высокоскоростных каналов информационного обмена в качестве базового инструмента межмодульной и межблочной связи.
- Использование однородной вычислительной среды и унифицированных модулей в различных системах КБО
- Комплексная обработка информации и гибкая реконфигурация при отказах
- Использование многоядерных процессоров

5

Требования к процессорам во встроенных системах

- Предсказуемость
- Энергопотребление
- Тепловыделение
- Надёжность

ARM Powered Products

ARM

- Парадигма программирования

  Набор инструкций

  Архитектура системы

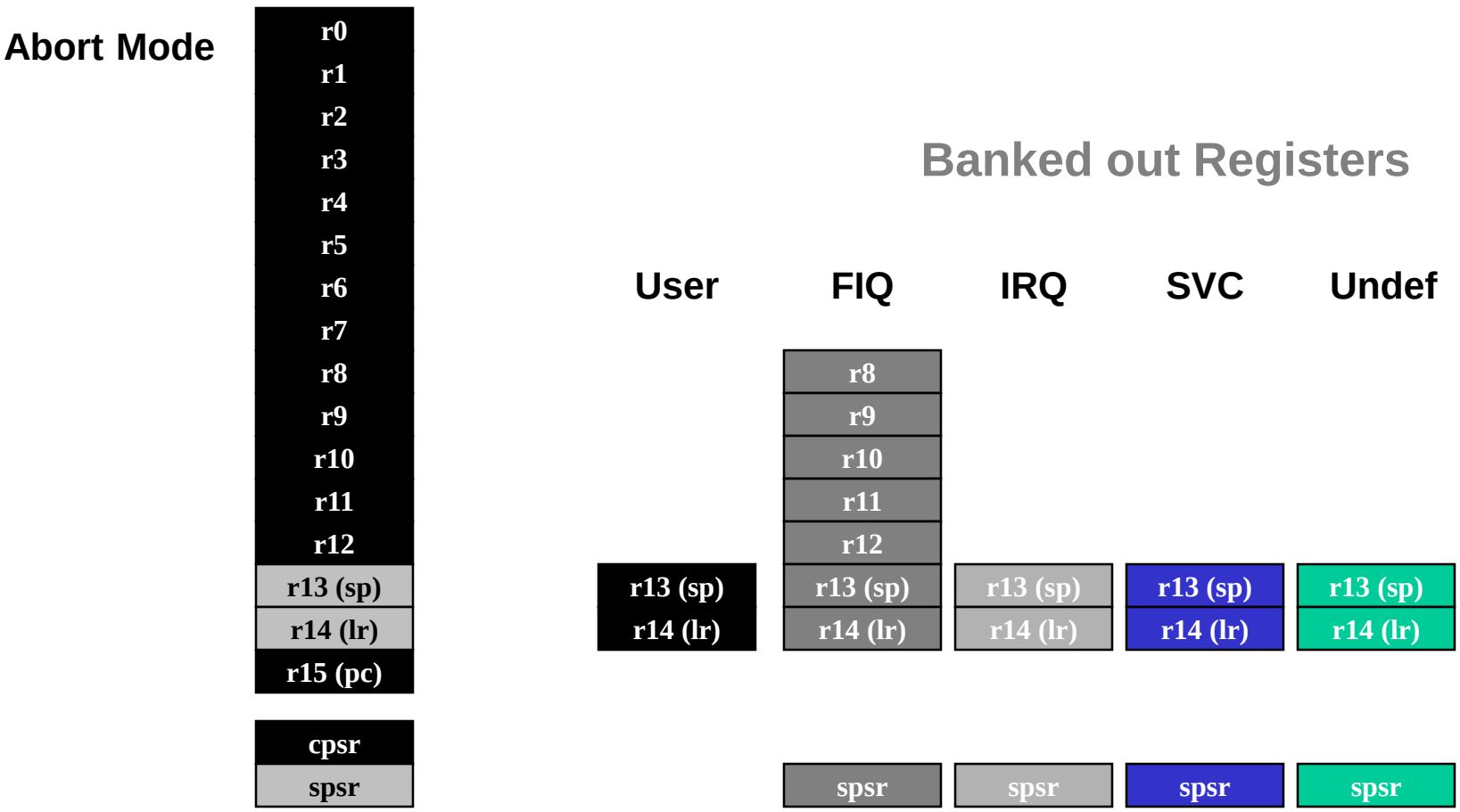
Размер типов данных и набор иструкций

- 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

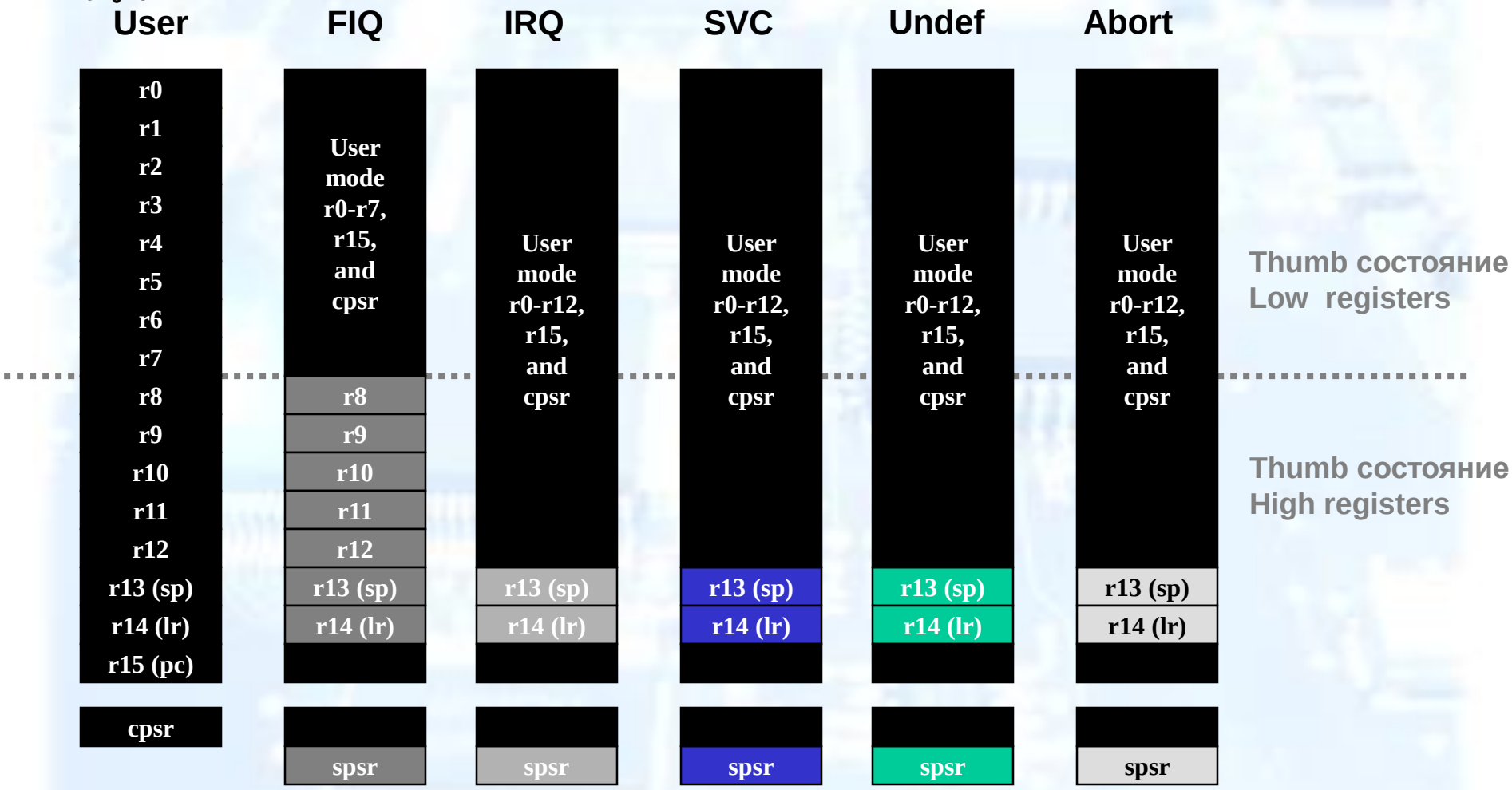
Режимы работы процессора

- 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

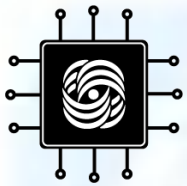
Набор регистров в ARM



Организация регистров



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



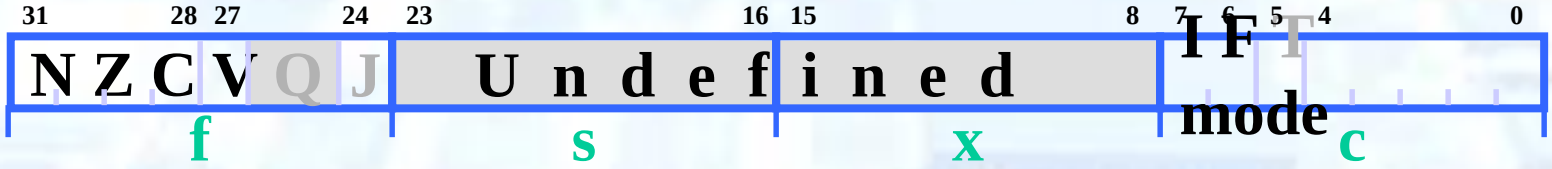
Типы регистров

- 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

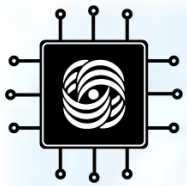
- В любом режиме работы процессора имеется доступ к следующим регистрам:
  - r0-r12 РОН
  - r13 (the stack pointer, sp) иr14 (the link register, lr)
  - program counter, r15 (pc)
  - current program status register, cpsr

  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

Процессор 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
```

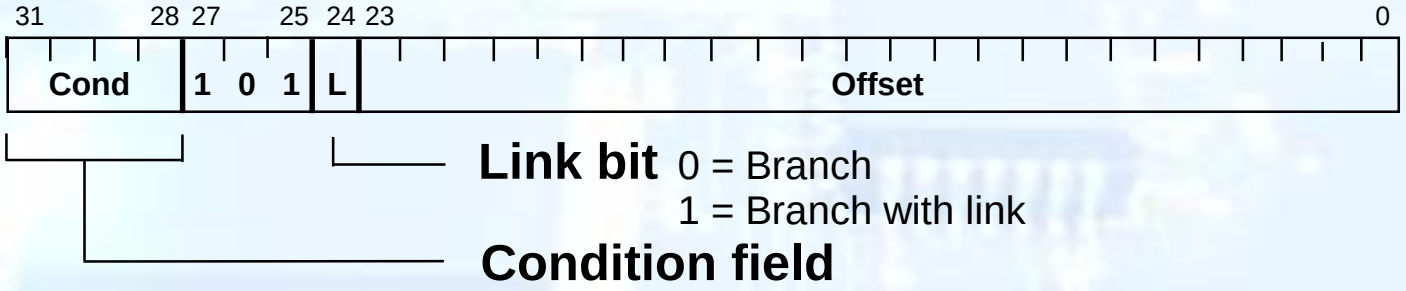
- Использование условных инструкций сравнения

```
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

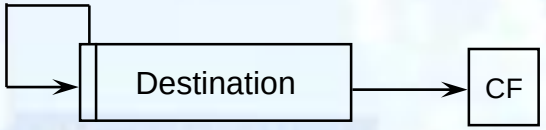
- Второй операнд отправляется на АЛУ через barrel shifter.

Barrel Shifter

LSL : Logical Left Shift

Умножение на 2

ASR: Arithmetic Right Shift

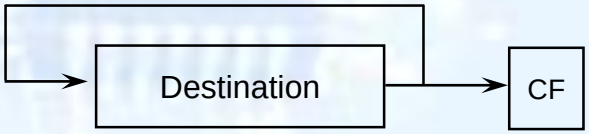


Деление на 2,
сохраняя бит флага

LSR : Logical Shift Right

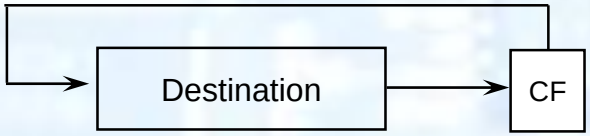
Деление на 2

ROR: Rotate Right



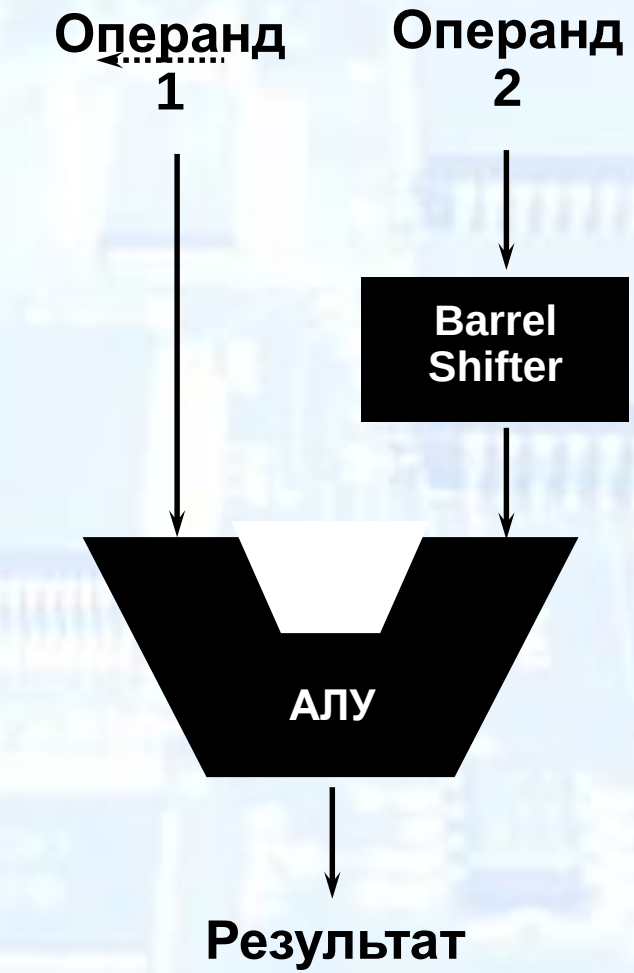
Циклическое смещение бита от LSB к MSB

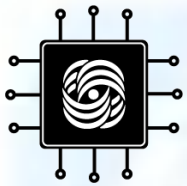
RRX: Rotate Right Extended



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

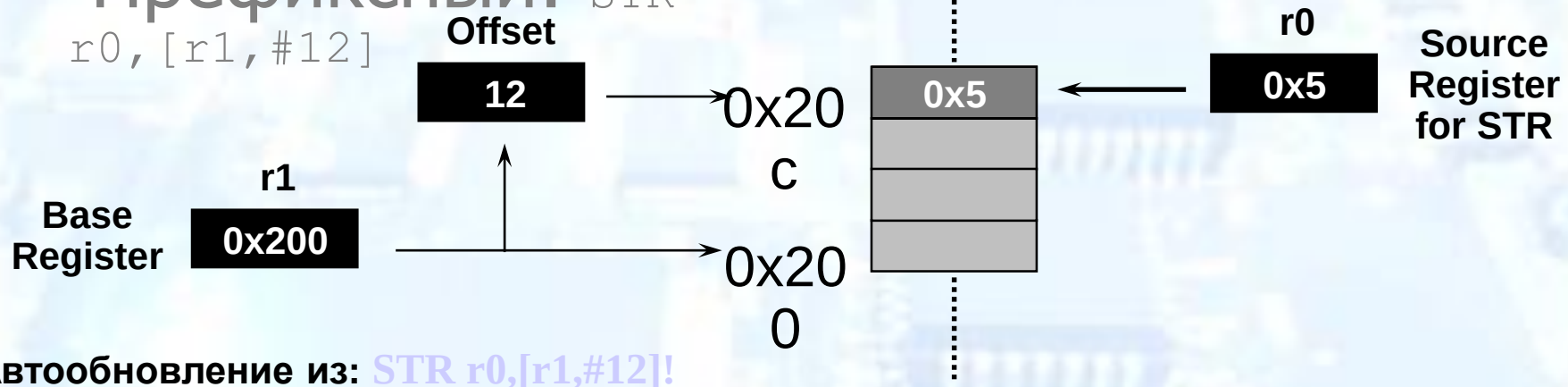
- Адрес доступные по LDR/STR определяется как значение регистра плюс смещение
- Для слова и беззнакового байта доступа, смещение может быть
  - 0 - 4095 bytes

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

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

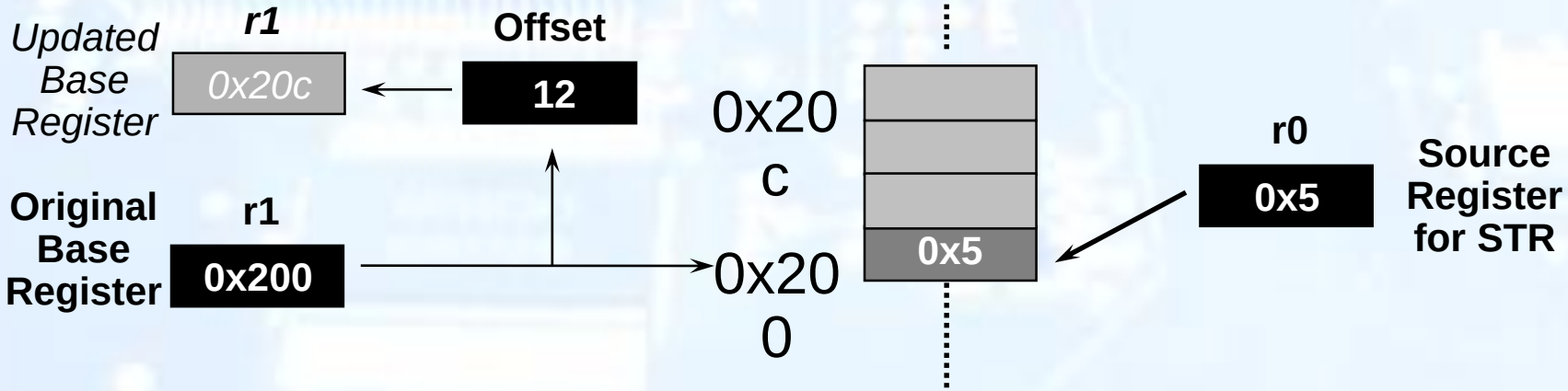
Префиксный или постфиксный адрес?

- 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:

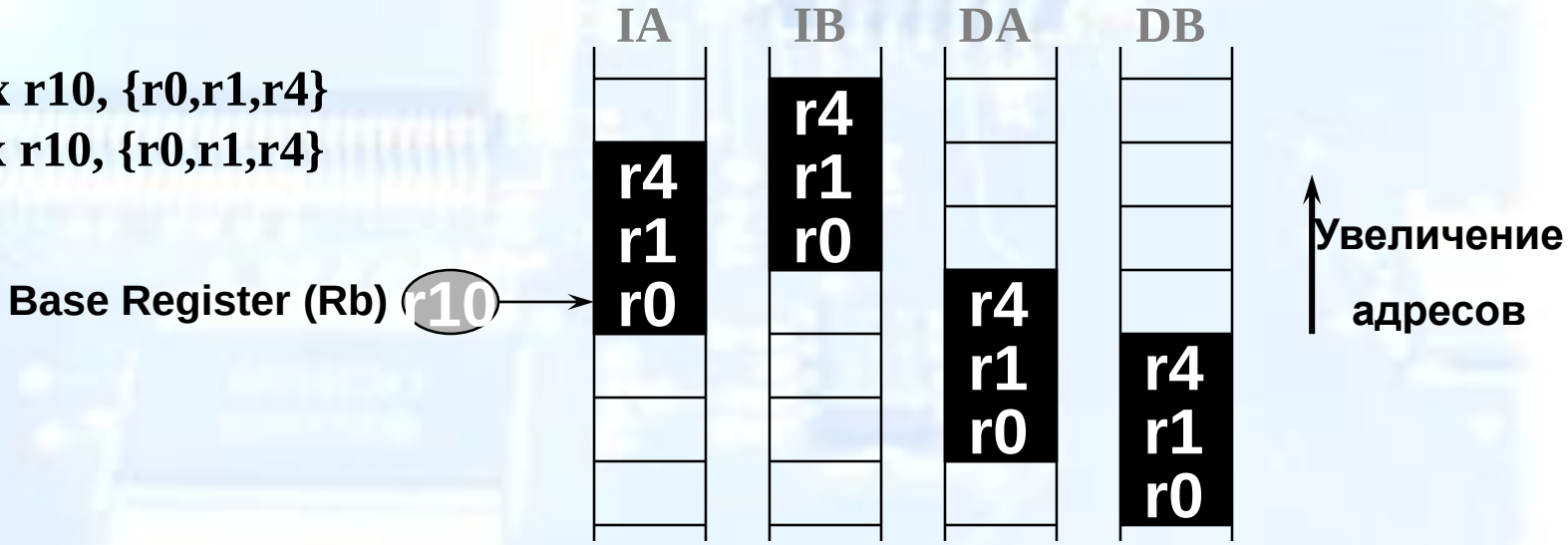
  LDMIA / STMIA инкрементить после

  LDMIB / STMIB инкрементить до

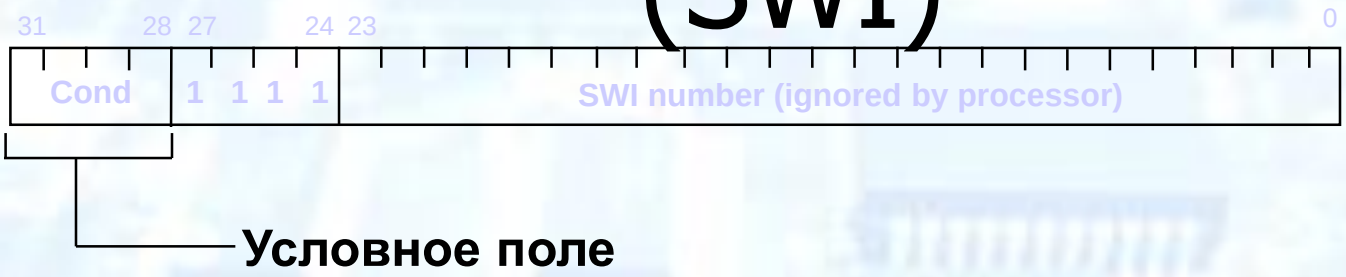
  LDMDA / STMDA декрементить после

  LDMDB / STMDB декрементить до

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

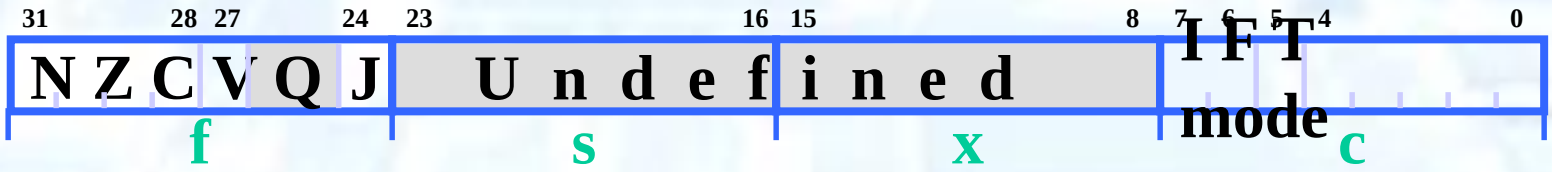


Программное прерывание (SWI)



- 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



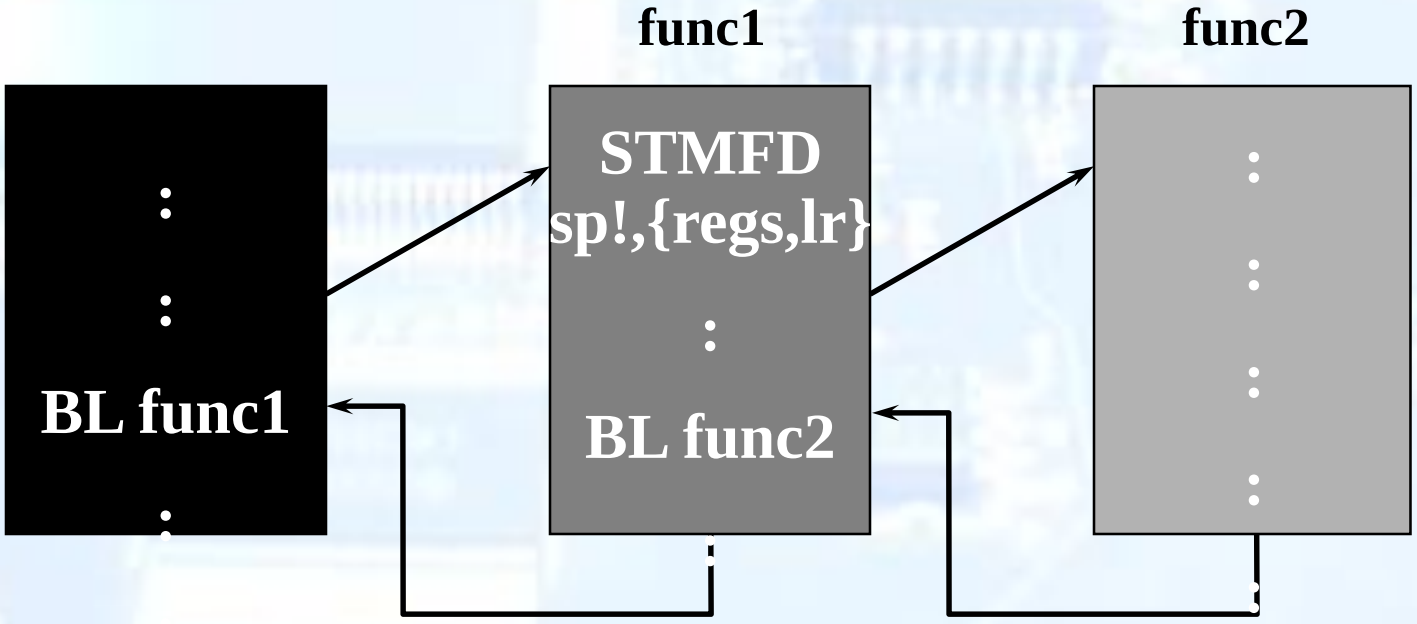
- MRS и MSR позволяет переместить содержимое CPSR / SPSR в или из регистра общего назначения
- Синтаксис:
  - `MRS{<cond>} Rd,<psr>  ; Rd = <psr>`
  - `MSR{<cond>} <psr[_fields]>,Rm  ; <psr[_fields]> = Rm`

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



Пример

```
While (i!=j)
    (i > j) ? i-=j : j-=i;
```

```
loop CMP Ri, Rj;
  SUBGT Ri, Ri, Rj ; i = i - j
  SUBLT Rj, Rj , Ri ; j = j – I
BNE loop          ; if “NE” (not equal),then loop
```

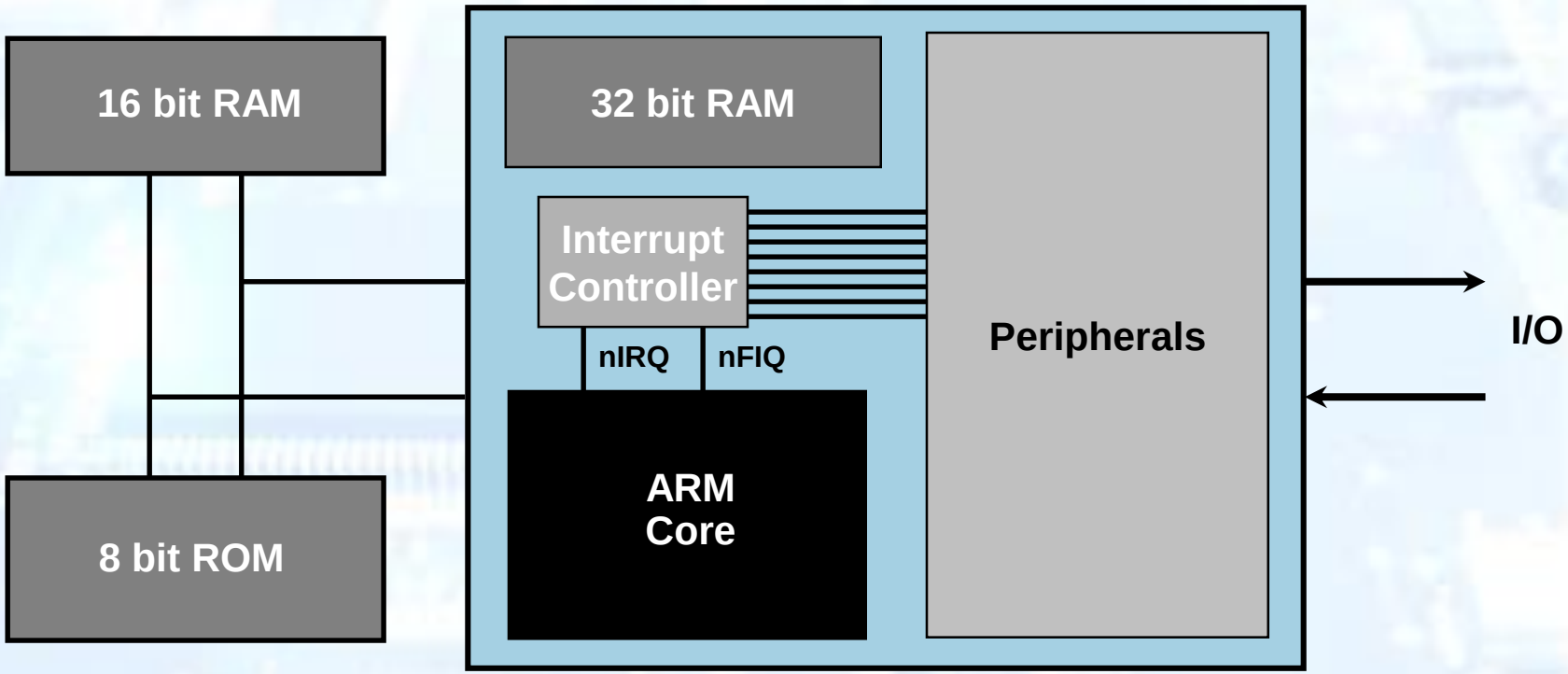
Процессор ARM

Парадигма программирования

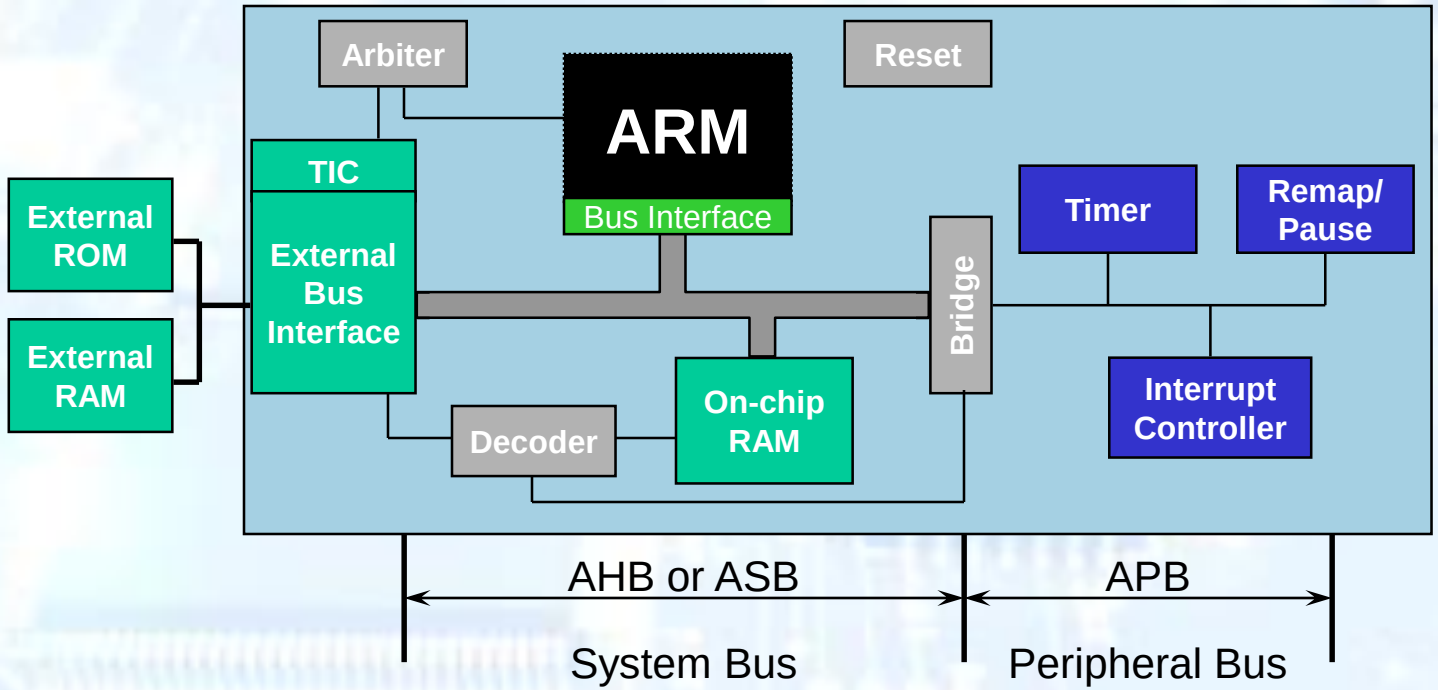
Набор инструкций

- 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

Процессоры DSP

- Особенности DSP процессоров
- Процессор NM6403

Особенности DSP процессоров

- Аппаратная поддержка программных циклов, кольцевых буферов
- Один или несколько операндов извлекаются из памяти в цикле исполнения команды
- Нет команд R,R->R
- Реализация однотактного умножения и команд, использующих в качестве операндов содержимое ячеек памяти

Особенности DSP процессоров (2)

- Сложение и умножение требуют:
  - произвести выборку двух операндов
  - выполнить сложение или умножение (обычно и то и другое)
  - сохранить результат или удерживать его до повторения
- Множественный доступ к памяти за один и тот же командный цикл.

Процессор NM6403

- 50 Mhz
- RISC ядро
  - 32-битные данные
  - 32-битные операции
  - 8 + 8 регистров
- Векторное устройство
  - Переменная разрядность
  - До 2048 параллельных умноженей

RISC-ядро

- 5-ти ступенчатый 32-разрядный конвейер;
- 32- и 64-разрядные команды (обычно выполняется две операции в одной команде);
- Два адресных генератора, адресное пространство - 16 GB;
- Два 64-разрядных программируемых интерфейса с SRAM/DRAM-разделяемой памятью;
- Формат данных - 32-разрядные целые;
- Регистры:
- 8 32-разрядных регистров общего назначения;
- 8 32-разрядных адресных регистров;
- Специальные регистры управления и состояния;
- Два высокоскоростных коммуникационных порта ввода/вывода,
- Аппаратно совместимых с портами TMS320C4x.

VECTOR-сопроцессор

- Переменная 1-64-разрядная длина векторных операндов и результатов;
- Формат данных - целые числа, упакованные в 64-разрядные блоки, в форме слов переменной длины от 1 до 64 разрядов каждое;
- Поддержка векторно-матричных и матрично-матричных операций;
- Два типа функций насыщения на кристалле;
- Три внутренних 32x64-разрядных RAM-блока

Производительность

- Скалярные операции:
  - 50 MIPS;
  - 200 MOPS для 32-разрядных данных;
- Векторные операции:
  - от 50 до 50.000+ MMAC (миллионов умножений с накоплением в секунду);
- I/O и интерфейсы с памятью:
  - пропускная способность двух 64-разрядных интерфейсов с памятью - до 800 Мбайт/сек;
- I/O коммуникационные порты - до 20 Мбайт/сек кажд

Особенности NM64003 (1)

- Возможность работы с входными сигналами (синапсами) и весами переменной разрядности (от 1 до 64 бит), задаваемой программно, что обеспечивает уникальную способность нейропроцессора увеличивать производительность с уменьшением разрядности операндов;
- Быстрая подкачка новых весов на фоне вычислений;
- (24 операции умножения с накоплением за один такт при длине операндов 8 бит);
- V аппаратная поддержка эмуляции нейросетей большой размерности;
- Реализация функции активации в виде пороговой функции или функции ограничения;

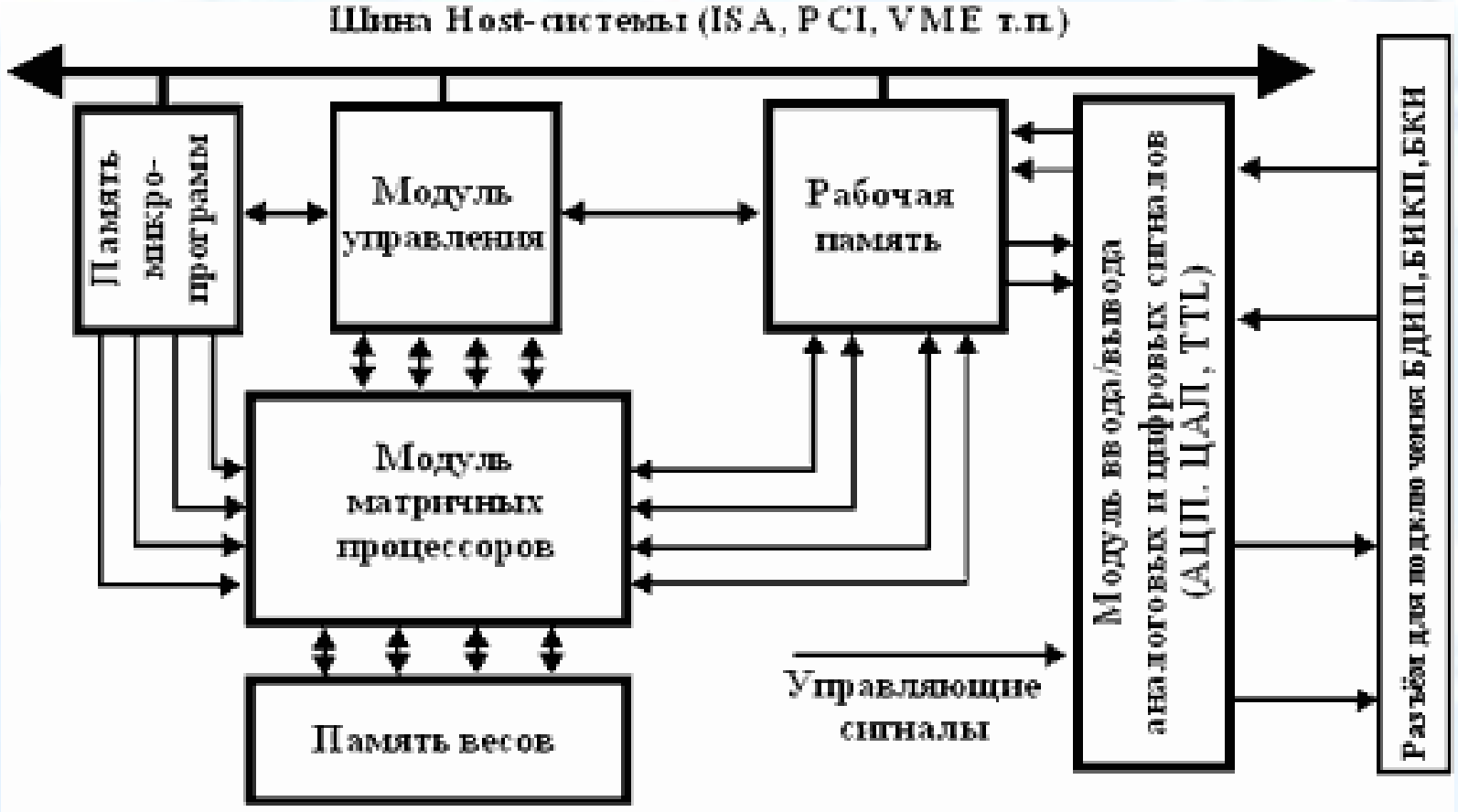
Особенности NM64003 (2)

- Наличие двух широких шин (по 64 разряда) для работы с внешней памятью любого типа: до 4Мб SRAM и до 16 Гб DRAM;
- Наличие двух байтовых коммуникационных портов ввода/вывода, аппаратно совместимых с коммуникационными портами TMS320C4x для реализации параллельных распределенных вычислительных систем большой производительности.
- Возможность работать с данными переменной разрядности по различным алгоритмам, реализуемым с помощью хранящихся во внешнем ОЗУ программ

Системы на NM 6403

- MC431 – однопроцессорная плата
- NM4 – четырехпроцессорная плата
- 6MCBO – 4 платы по 6 процессоров и платы для обработки входных сигналов

Схема нейровычислителя



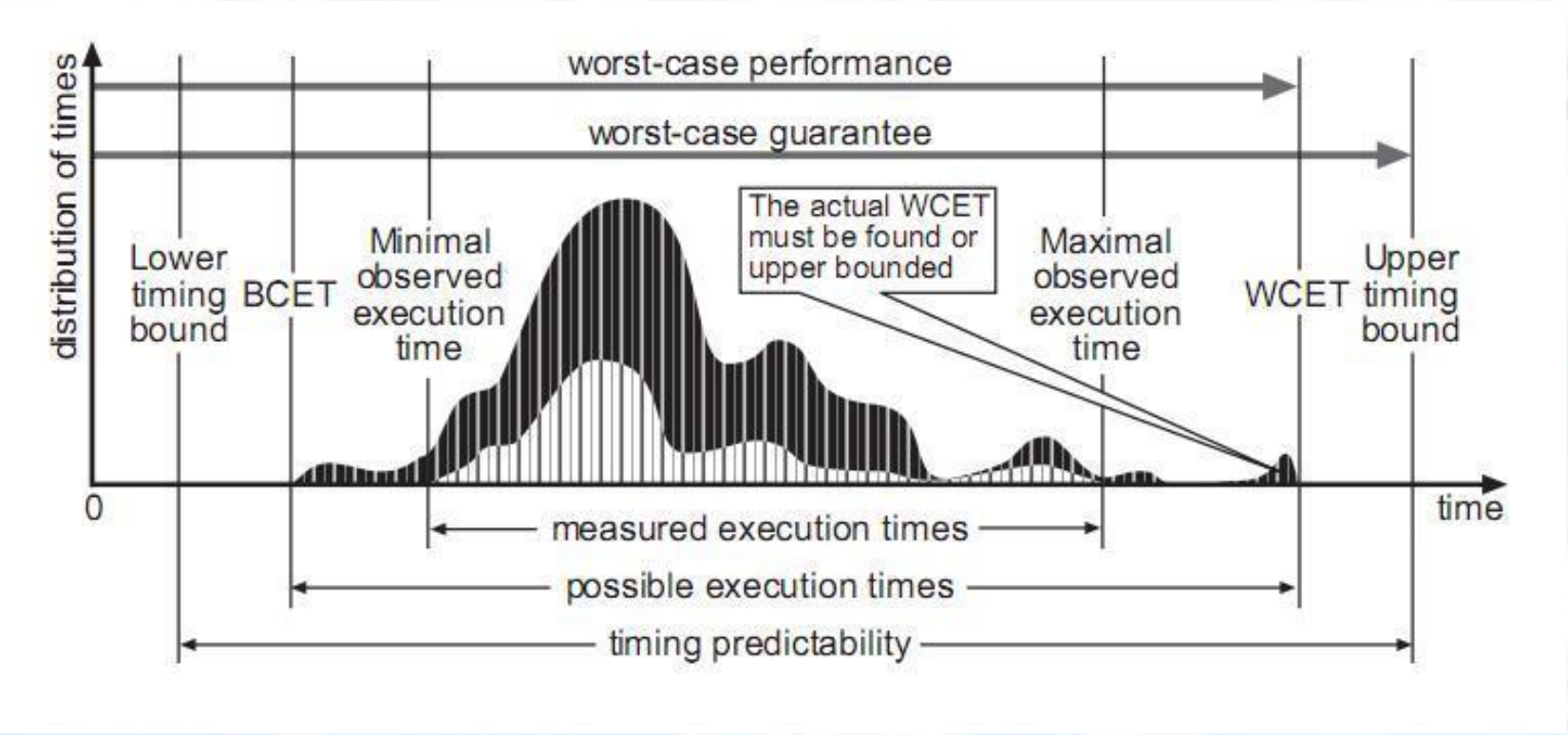
WCET

- Постановка задачи
- Статический метод
- Измерительный метод
- Гибридный метод
- Метод Bound Checking

Введение

- WCET – задача оценки наихудшего времени выполнения программ
- Актуальна при проектировании и анализе программ для РВ систем
- В РВ системах должно быть гарантировано время реакции системы на внешние события

WCET

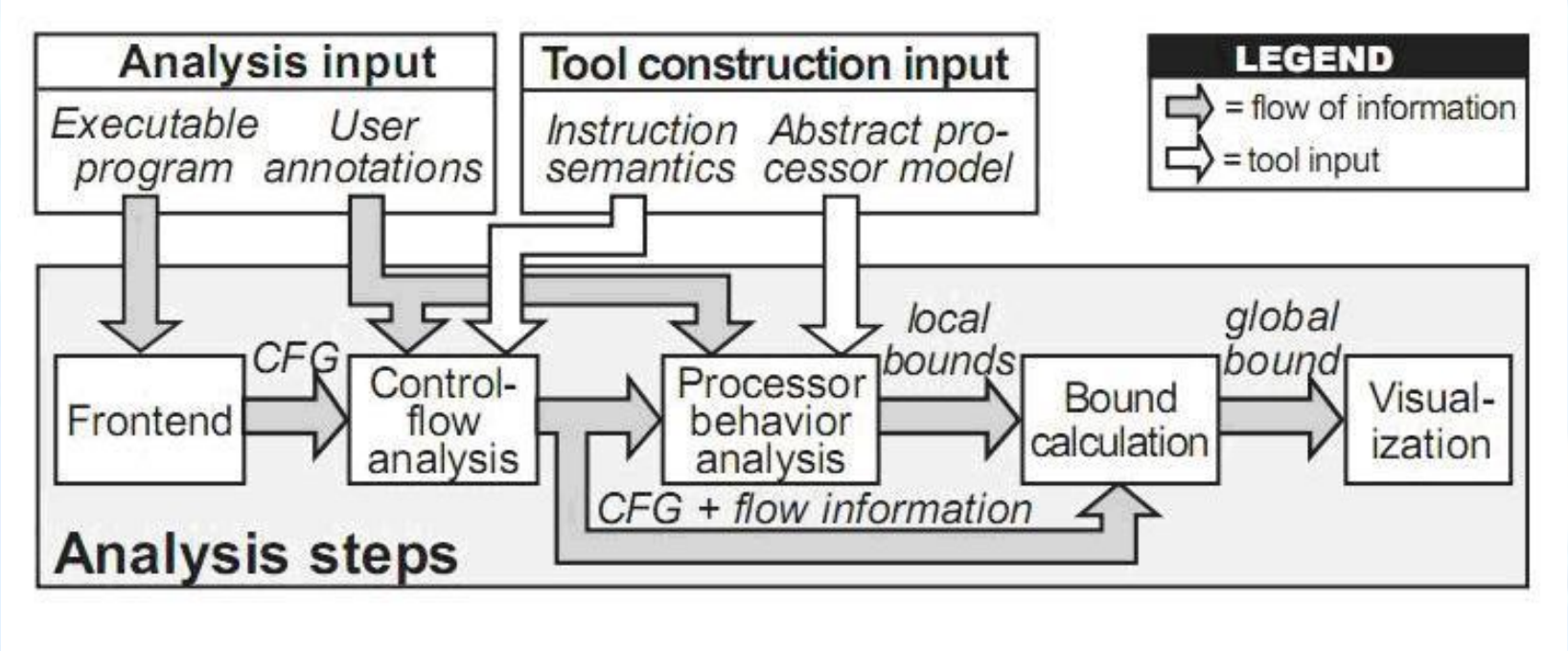

distribution of times
worst-case performance
worst-case guarantee
Lower timing bound
BCET
Minimal observed execution time
The actual WCET must be found or upper bounded
Maximal observed execution time
WCET
Upper timing bound
0
time
measured execution times
possible execution times
timing predictability

Статический метод

- Анализ программы без выполнения на реальном оборудовании
- Общая схема:
  - Построение CFG
  - Анализ линейных участков
  - Вычисление оценки WCET по CFG

Статический метод

- Пример организации:

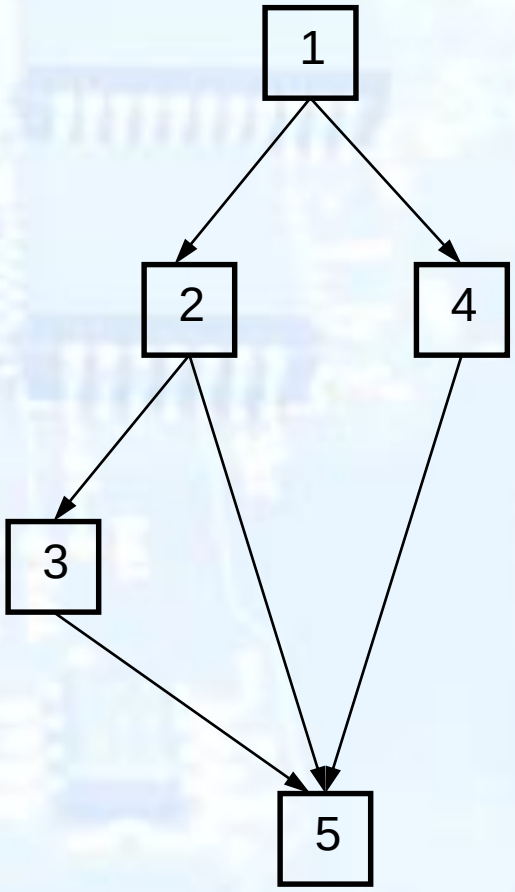


CFG

- CFG – граф:
  - вершины - линейные участки программы
  - ребра – возможные переходы

CFG

```
...                 // 1
if (a>5)
{
    sleep(5);       // 2
    if (a>7)
    {
        sleep(7);   // 3
    }
}
else
{
    sleep(7);       // 4
}
sleep(5);           // 5
```

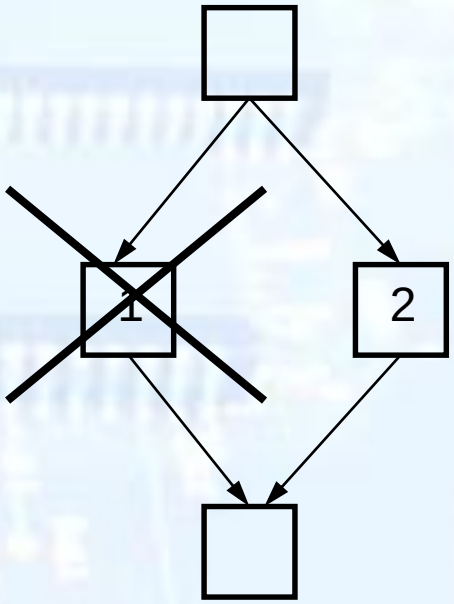


Анализ потоков управления

- Цель:
  - Вычисление количества итераций циклов
  - Выявление границ рекурсий
  - Определение недостижимых блоков

Анализ потоков управления

```
if (a>10)
{
    if (a<5)
    {
        …   // 1
    }
    else
    {
        …   // 2
    }
}
```



- Блок 1 - недостижим

Анализ поведения процессора

- Вычисление времени выполнения линейных участков
- При этом учитывается влияние архитектурных компонент:
  - Кэши
  - Конвееры
  - Предсказатель переходов

Анализ поведения процессора

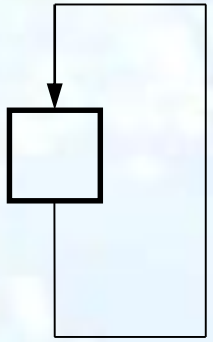
Пример: анализ поведения кэша:

```
for ( i=0 ; i<3 ;
   i++)
{
   a+=i;
}
```

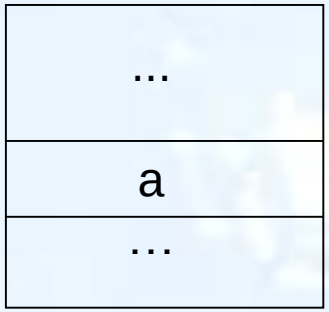
1 проход: кэш – промах, time = 10

2 проход: кэш – попадание, time = 7

3 проход: кэш – попадание, time = 7



кэш:



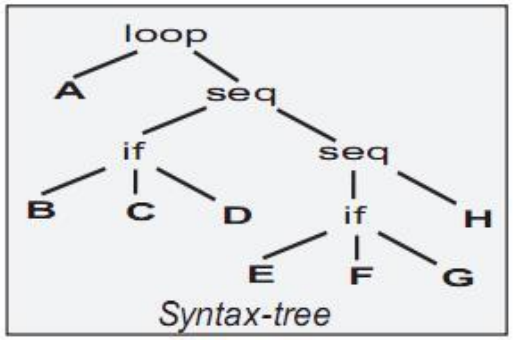
Вычисление оценки WCET

- Производится проход по графу и вычисление оценки WCET
- Существует три подхода к вычислению оценки:
  - structure-based
  - path-based
  - implicit path enumeration techniques (IPET)

Structure-based

- Перебираются листья синтаксического дерева
- Выделяются группы листьев и объединяются по определенным правилам
- После полного прохода получается одна вершина
- WCET = время выполнения этой вершины

Structure-based

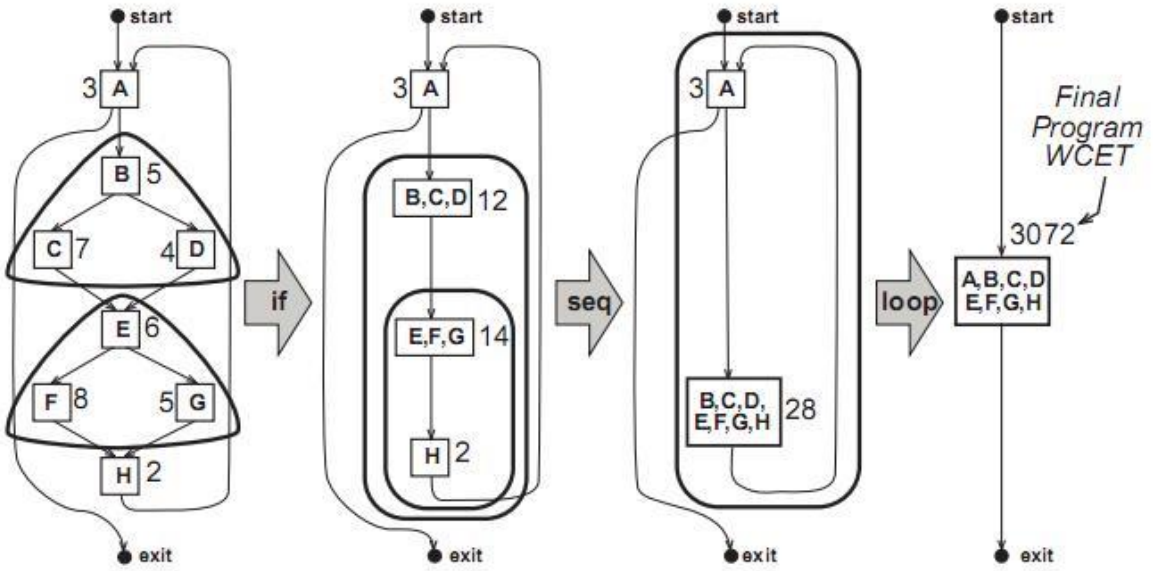


T(seq(S1,S2)) = T(S1) +T(S2)

T(if(Exp) S1 else S2) =
T(Exp) + max(T(S1),T(S2))

T(loop(Exp,Body)) =
T(Exp) +
(T(Exp) +T(Body)) * (maxiter-1)

Transformation rules

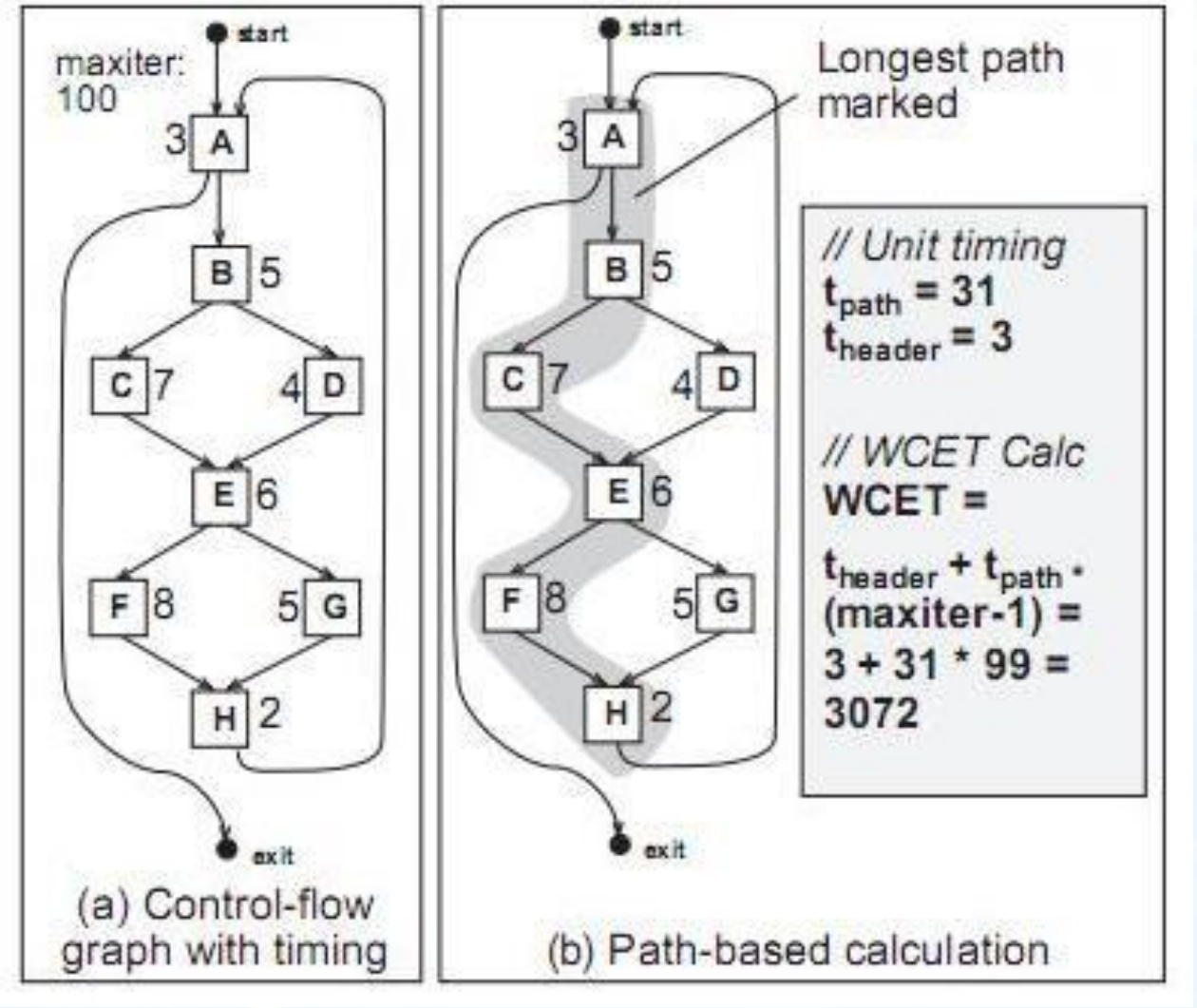


(d) Structure-based calculation

Path-based

- Перебираются все возможные пути программы
- Для каждого пути вычисляется время выполнения
- Выбирается путь с максимальным временем выполнения
- WCET = время этого пути

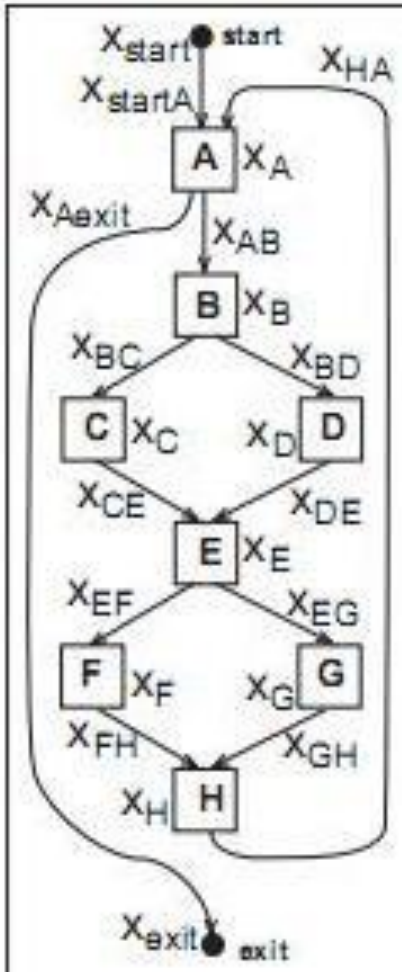
Path-based


maxiter:
100
start
3 A
B 5
C 7
4 D
E 6
F 8
5 G
H 2
exit
(a) Control-flow graph with timing
start
3 A
B 5
C 7
4 D
E 6
F 8
5 G
H 2
exit
Longest path marked
// Unit timing
t_{path} = 31
t_{header} = 3
// WCET Calc
WCET =
$t_{header} + t_{path} \cdot$ (maxiter-1) =
3 + 31 * 99 =
3072
(b) Path-based calculation

IPET

- Всем ребрам и вершинам сопоставляется коэффициент - число проходов по ребру или вершине
- Составляется система уравнений следующего вида:
  - Коэффициент вершины = сумма коэффициентов входящих ребер
  - Коэффициент вершины = сумма коэффициентов исходящих ребер
- Составляется функция, равная сумме произведений коэффициентов на время выполнения каждого линейного участка
- WCET – максимальное значение этой функции

IPET



// Start and exit constraints

$x_{start} = 1, x_{exit} = 1$

// Structural constraints

$x_{start} = x_{startA}$

$x_A = x_{startA} + x_{HA} = x_{Aexit} + x_{AB}$

$x_B = x_{AB} = x_{BC} + x_{BD}$

$x_C = x_{BC} = x_{CE}$

. . .

$x_H = x_{FH} + x_{GH} = x_{HA}$

$x_{exit} = x_{Aexit}$

// Loopbound constraint

$x_A \leq 100$

// WCET Expression

$WCET = \max(x_A*3 + x_B*5 + x_C*7 + \ldots + x_H*2) =$

$= 3072$

(c) IPET calculation

Измерительный метод

- Программа выполняется на реальном оборудовании
- Производится серия запусков программы на различных наборах подготовленных входных данных
- Определяется оценка наихудшего времени выполнения

Измерительный метод

- Преимущества:
  - Не требуется описание модели вычислителя
- Недостатки:
  - Необходимость запуска программы на целевом оборудовании
  - Подготовка входных данных, покрывающих все возможные пути выполнения
  - Оценка WCET не точна

Гибридный метод

- Соединяет в себе измерительный и статический методы
- Общая схема анализа:
  - Построение CFG
  - Вычисление времени выполнения каждого линейного участка измерительным методом
  - Вычисление оценки wcet по графу статическим методом

Гибридный метод

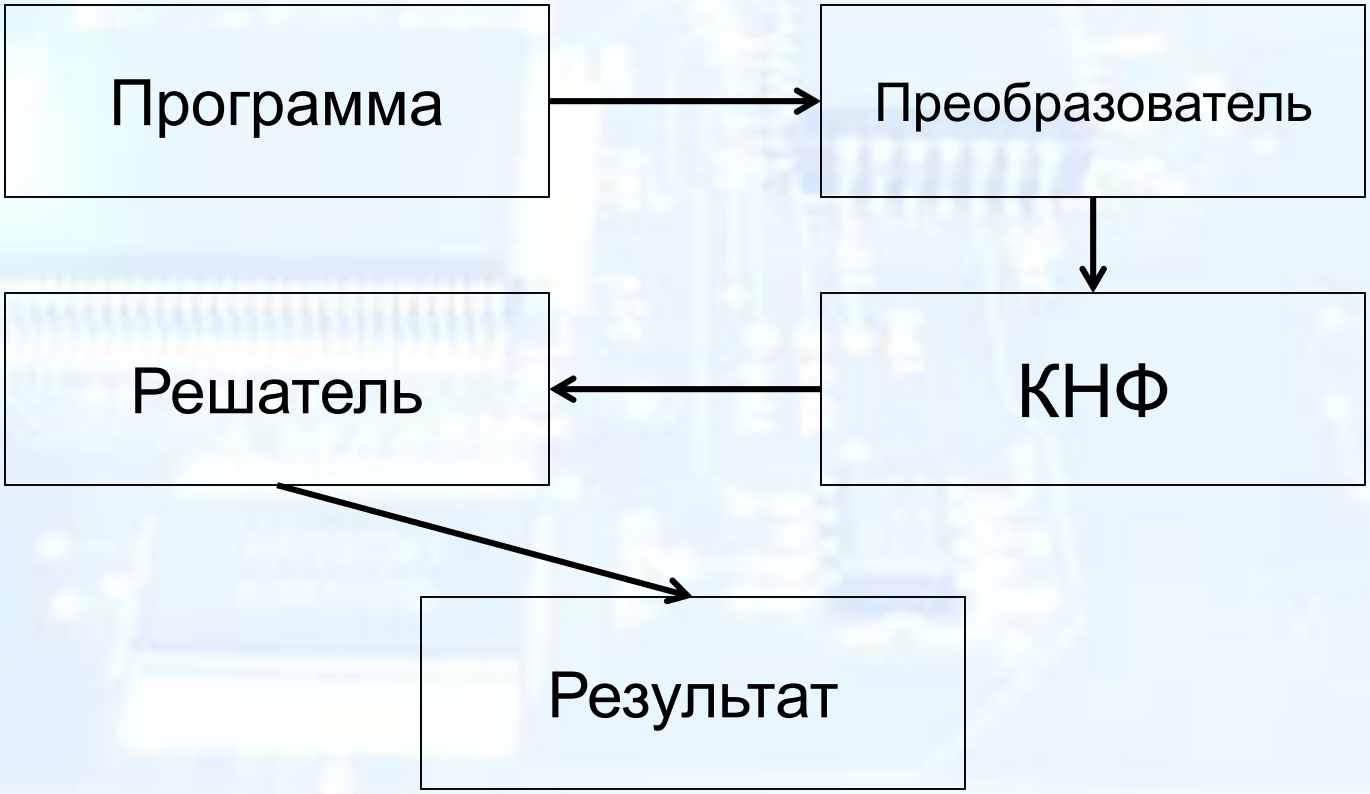
- Преимущества:
  - Сокращается количество выполнений на реальном оборудовании
  - Не требуется подготовка входных данных
- Недостатки:
  - Необходимость запуска на целевом вычислителе
  - Оценка WCET завышена

Bounded model checking

- Технология верификации программ на модели с ограничением на длину вычислений
- Основана на представлении программ в виде КНФ
- Проверка свойств производится путем проверки выполнимости построенной КНФ

Bounded model checking

- Типичная схема анализа:



Bounded model checking

- Построение КНФ:

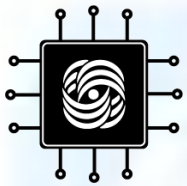
```
…
if ( a < 5 )
      a = b + 1;
else
      a = 5;
return a;
```

КНФ:

(a_1 = b_0 + 1) &&
(a_2 = 5) &&
[(a_0 < 5 &&
a_3 = a_1) ||
(a_0 >= 5 &&
a_3=a_2)]
&& (out = a_3)

Bounded model checking

- Циклы:
  - Ограничение количества итераций мажорантой
  - Разворачивание циклов



Bounded model checking

```
for ( int i = 0; i < 3; ++ i
    )
{
    if ( a < 5 )
        a += i;
}
```

```
i = 0;
if ( i < 3 )
{
    if ( a < 5 )
        a += i;
    ++i;
    if ( i < 3 )
    {
        if ( a < 5 )
            a += i;
        ++i;
...
```

Bounded model checking

- Проверка свойств:

```
void f(int a, int b)
{
   if ( a < 5 )
        a = b + 1;
   else
        a = 5;
   assert( a <= 5 );
   return a;
}
```

КНФ:

(Исходная КНФ программы) *&&*

!(a <= 5)

Если полученная КНФ выполнима, то свойство не выполняется!

Bounded model checking

- Проверка выполнимости КНФ – решение задачи ВЫП
- Решатели:
  - Z3 (Microsoft)
  - Yices (SRI)
  - Boolector

Bounded model checking

- Ограничение на длину вычислений:
  - Построение КНФ длины не более *k*
  - Проверка свойств полученной КНФ
  - Если свойство выполняется,
    то увеличение *k*
  - Позволяет быстро проверять свойства!

Использование BMC для оценки WCET

- Известно время выполнения каждого линейного участка
- Добавление вычисления времени выполнения в КНФ
- assert (time <= wcet); в конце программы
- запуск проверки, коррекция wcet, пока свойство нарушается

Использование BMC для оценки WCET

Исходная программа

- void f(int a) {
  a = a + 5; // time = [1, 2]
  if (a < 0) {
  /* path1 */ // time = [10, 15]
  }
  /* path2 */ // time = [5, 16]
  }
- при a принадлежащем [-4; 0]

Использование BMC для оценки WCET

```
void f(int a) {
    a = a + 5; // time = [1,
    2]
        if (a < 0) {
            /* path1 */
          // time = [10, 15]
        }
        /* path2 */
        // time = [5, 16]
    assert(time <= 0)
     }
```

КНФ:

time_0 = 0 &&

(-4 <= a_0 <= 0) &&

…

time_1 = time_0 + [1,2] &&

…

time_2 = time_1 + [10,15] &&

a<0 && time_3 = time_2 ||

!(a<0) && time_3 = time_1

time_4 = time_3 + [5,16] &&

*&& **!(time_4 <= 0)***

Использование BMC для оценки WCET

- Система выполнима
- Строим модель, time_3 = 16
- wcet = 16
- Строим КНФ заново:

  Исходная *КНФ &&* ***!(time_3 <= 16)***

Использование BMC для оценки WCET

- После нескольких итераций получаем wcet = 33

КНФ && !(time_3 <= 33)

- Система не выполнима
- Наихудшее время выполнения найдено

Литература

- Материалы сайта www.arm.com
- “32-разрядные высокопроизводительные RISC-процессоры семейства ARM” (http://www.gaw.ru/html.cgi/txt/doc/micros/arm/arh/index.htm)
- Нейропроцессор NM6403. Материалы компании ЗАО НТЦ «Модуль». Адрес в Интернет: http://www.module.ru
- Черников В. М., Виксне П. Е., Фомин Д. В., Шевченко П. А. "Архитектурные особенности нейропроцессора NM6403". Всероссийская научно-техническая конференция "Нейроинформатика-99". Сборник научных трудов. Часть 2, Москва, 1999, С.93-101. (http://library.mephi.ru/data/scientific-sessions/1999/Neuro_2/093.pdf)
- Wilhelm R. et al. The worst-case execution-time problem—overview of methods and survey of tools //ACM Transactions on Embedded Computing Systems (TECS). – 2008. – Т. 7. – №. 3. – С. 36. (http://moss.csc.ncsu.edu/~mueller/ftp/pub/mueller/papers/1257.pdf)

Спасибо за внимание!