# Введение

Данный документ является отчётом о проделанной работе по проекту MEPHI-CPU 2015: разработка специализированного процессора собственной архитектуры схожей с архитектурой MIPS. Проект проектировался для загрузки на ПЛИС семейства Spartan-6 фирмы Xilinx, находящейся на плате Atlys фирмы Digilent. Кроме ПЛИС, предполагалось использование оперативной платы DDR-2, распаянной на плате, для хранение пользовательских программ, и USB-UART мост для передачи пользовательской информации от ПК к процессору и обратно.

Специализация процессора заключается в возможности вычисления хеш-сумм, согласно алгоритмам MD4, MD5, SHA-1 и т.п.

# Архитектура системы команд

# Ассемблер

Ассемблер (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) assembler — сборщик) — транслятор исходного текста программы, написанной на [языке ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0), в программу на [машинном языке](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Язык ассемблера ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) assembly language) — машинно-ориентированный [язык низкого уровня](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) с командами, один к одному соответствующими командам [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80). Фактически, они и представляют собой более удобную для человека символьную форму записи команд и их [аргументов](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) — мнемокоды .

Кроме того, язык ассемблера позволяет использовать символические [метки](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) вместо адресов ячеек памяти, которые при ассемблировании заменяются на вычисляемые ассемблером [относительные адреса](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D1%80%D0%B5%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)#.D0.BE.D1.82.D0.BD.D0.BE.D1.81.D0.B8.D1.82.D0.B5.D0.BB.D1.8C.D0.BD.D1.8B.D0.B9), а также так называемые [директивы](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%80%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D1%8B_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0) (команды ассемблера, не переводимые в машинные команды процессора, а выполняемые самим ассемблером).Директивы ассемблера позволяют, в частности, включать блоки данных, задавать новые имена регистрам, присваивать значения именам.

Для написания ассемблера для данного процессора использовался язык программирования Python 3.4. В программе реализован перевод 16 инструкций R и M типа, реализована работа с 4 различными директивами. В результате работы программы создаётся бинарный файл, содержащий машинный код программы на ассемблере, так же создаётся бинарный файл для записи кода в память процессора.

Ассемблер обрабатывает инструкции указанные в таблице №1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица №1. Формат инструкции (всего 16 бит) | | | | | | | | | | |
| Описание | | Ассемблерный код | кодировка  (4 бит) | | | | некоторые аргументы инструкции (12 бит) | | | |
| Сложение | R | ADD rd rs rt | 0 | 0 | 0 | 0 | rs | | rt | rd |
| Сложение с константой | R | ADDI rd rt imm | 0 | 0 | 0 | 1 | imm | | rt | rd |
| Побитовое ИЛИ | R | OR rd rs rt | 0 | 0 | 1 | 0 | rs | | rt | rd |
| Побитовое И | R | AND rd rs rt | 0 | 0 | 1 | 1 | rs | | rt | rd |
| Побитовое исключающее ИЛИ | R | XOR rd rs rt | 0 | 1 | 0 | 0 | rs | | rt | rd |
| Побитовое ИЛИ-НЕ | R | NOR rd rs rt | 0 | 1 | 0 | 1 | rs | | rt | rd |
| Логический сдвиг влево | R | SLL rd rs rt | 0 | 1 | 1 | 0 | rs | | rt | rd |
| Циклический сдвиг влево | R | ROT rd rs rt | 0 | 1 | 1 | 1 | rs | | rt | rd |
| Условное ветвление, если не равны | R | BNE rd rs rt | 1 | 0 | 0 | 0 | rs | | rt | rd |
| загрузка из памяти в регистр | M | LDL addr | 1 | 0 | 0 | 1 | 0 | \* | addr | |
| M | LDH addr | 1 | 0 | 0 | 1 | 1 | \* | addr | |
| сохранение из регистра в память | M | STL addr | 1 | 0 | 1 | 0 | 0 | \* | addr | |
| M | STH addr | 1 | 0 | 1 | 0 | 1 | \* | addr | |
| безусловный переход по адресу | M | JMP addr | 1 | 0 | 1 | 1 | 0 | 0 | addr | |
| вызов подпрограммы по адресу | M | JAL addr | 1 | 0 | 1 | 1 | 0 | 1 | addr | |
| безусловны переход по значению регистра | M | JR rs | 1 | 0 | 1 | 1 | 1 | 0 | rs | |
| вызов подпрограммы по значению регистра | M | JALR rs | 1 | 0 | 1 | 1 | 1 | 1 | rs | |
| Нет операции |  | NOP | 1 | 1 | 0 | 0 |  | | | |

Данный ассемблер поддерживает следующие директивы:

* директива с блоком данных .data;
* директива с командами .text;
* директива .def для изменения имени регистра;
* директива .set для присвоения имени константе.

Пример кода на языке ассемблер:

.text

add $a, out, $zero # Складываем значение регистров $3 (out) и $zero

addi $1, $2, p1 # Прибавляем к регистру $2 значение p1 = 8 (.set)

ld res # Загружаем в MR значение 455 (res из .data)

.data

res 455

.text

addi $MR, $MR, sum # MR + 15 (sum = 15 из .set) теперь в MR число 470

st res # сохраняем в адрес памяти sum число 470

nop (5)

.set

p1 8

sum 0b1111

.def

out $3

# Общее описание системы

Проектируемая система состоит из нескольких основных частей (рисунок 1):

* CPU – специализированное процессорное ядро, собственной разработки;
* MC – контроллер доступа в память (далее более подробно);
* IOHub - модуль периферии собственной разработки, отвечающий за взаимодействием с ПК;
* DDR2 – оперативная память, представленная микросхемой на плате и подключённая к ПЛИС;
* USB-UART Bridge – мост USB-UART, представленный микросхемой на плате и подключённый к ПЛИС.

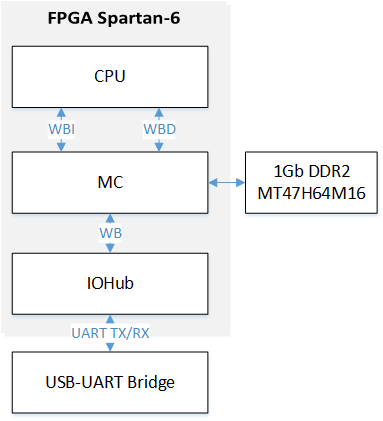
****

Рисунок 1. Блок-схема разрабатываемой СнК

В ходе разработки микроархитектуры было принято решение использовать стандартный интерфейс Wishbone для взаимодействия основных частей СнК.

Описать WBI, WBD, WB, UART TX/RX

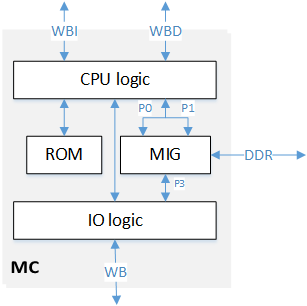


Рисунок 2. Блок-схема разрабатываемого контроллера доступа в память

# Описание блоков ядра процессора

Таблица 1. Системные сигналы

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Направление | Разрядность |
| sys\_clk | Тактовый сигнал | IN | 1б |
| sys\_rst | Сигнал сброса | IN | 1б |

Таблица 2. Интерфейс инструкций

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Направление | Разрядность |
| instr\_stb\_o | Ведомый интерфейс работает, если сигнал равен 1 | OUT | 1 |
| instr\_we\_o | Разрешение записи адреса | OUT | 1 |
| instr\_addr\_o | Адрес новой инструкции | OUT | 16 |
| instr\_ack\_i | Оповестительный сигнал – ведомое устройство подтверждает достоверность инструкции | IN | 1 |
| instr\_data\_i | Инструкция | IN | 16 |

Таблица 3. Интерфейс данных

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Назначение | Направление | Разрядность |
| data\_stb\_o | Ведомый интерфейс работает, если сигнал равен 1 | OUT | 1 |
| data\_we\_o | Разрешение записи данных | OUT | 1 |
| data\_addr\_o | Адрес записи данных | OUT | 16 |
| data\_data\_i | Входные данные | IN | 16 |
| data\_data\_o | Выходные данные | OUT | 16 |
| data\_ack\_i | Оповестительный сигнал – ведомое устройство подтверждает достоверность данных | IN | 1 |

# Описание контроллера доступа в память

**Карта памяти**

. . .

. . .

addr [9:0] (согласно ISA)

ROM

RAM

I/O

**0**0 000000000

abs\_addr [10:0]

abs\_addr [10] – признак принадлежности адреса (1 – кRAM, 0 – к ROMили к I/O).

0**1** 000000000

01 11111111

1 0000000000

1 1111111111

. . .

abs\_addr [9] – признак принадлежности адреса (1 – кI/O,

0 – к ROM). Рассмотрение этого признака не имеет смысла для RAM.

0x000

0x00C

0x200

0x3FF

0x400

Максимум 512 инструкций по 16 бит. Один адрес относится к одному байту в оперативной памяти.

00 000001100

7 инструкций по 16 бит.

0x7FF

**Начальная загрузка**

Процессор начинает работу (PC перекидывается в область RAM) при условии, что в первом регистре, принадлежащим области I/Oзагружены все единицы. Наличие там единиц означает, что все данные по UARTзагружены в ОЗУ.

**Ассемблерный код, записанный в ROM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адрес PC | | | Ассемблерный код | Описание |
| DEC | HEX | BIN |
| 0 | 00 | 0000 | NOR t0, zero, zero | Заполнение регистра t0 всеми единицами |
| 2 | 02 | 0110 | ADDI t1, zero, 0100 | Присвоение регистру t1 значения «4» длядальнейшего его использования приизменении PC (PC→PC+4 –перескок через строку)а также для перехода по адресу «4» (PC→4) во время ветвления |
| 4 | 04 | 0010 | LDL 0x200\* | Загрузка в регистр MR (L) из памяти по адресу 0x200\* |
| 6 | 06 | 0100 | LDH 0x200\* | Загрузка в регистр MR (H) из памяти по адресу 0x200\* |
| 8 | 08 | 1000 | BNE t1, t0, MR | При невыполнении условия равенства t0 и MR, PC→PC+t1,т.е. +4 |
| 10 | 0A | 1010 | JMP 0x200\* (здесь addr[9:0] = 000000000) | Переход PC поадресу 0x200\* (переход PC к первому адресу RAM)JMP позволяет перемещаться только вRAM или по RAM! (см. комментарий) |
| 12 | 0C | 1100 | JR t1 | Переход PC по адресу, записанному в t1=4 (т.е. в этом цикле переходак RAM не будет. PCостанется в ROM). |

\*Звездочкой отмечены абсолютные адреса (abs\_addr)

Комментарий

*По поводу JMP и JAL. Это единственные инструкции, которые имеют фиксированный девятый бит регистраabs\_addr [9], равный для JMP – 0, а для JAL – 1. Так как вся программа будет находиться в RAM, то с JAL все хорошо. Ветвимся по RAM. С JMP ситуация хуже. Тут Даниэлю придется передавать в логику, связывающую память и процессор abs\_addr с девятым битом, равным единице, а не нулю. В итоге, указанный в ISAaddr, будет являться локальным адресом в RAM при использовании JMP и JAL. Для остальных инструкций типа М адрес будет абсолютным.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| загрузка  из памяти  в регистр | MR[15:0]MEM[abs\_addr] | LDL abs\_addr | 1 | 0 | 0 | 1 | 0 | abs\_addr | |
| MR[31:16]MEM[abs\_addr] | LDH abs\_addr | 1 | 0 | 0 | 1 | 1 | abs\_addr | |
| сохранение  из регистра  в память | MEM[abs\_addr]MR[15:0] | STL abs\_addr | 1 | 0 | 1 | 0 | 0 | abs\_addr | |
| MEM[abs\_addr]MR[31:16] | STH abs\_addr | 1 | 0 | 1 | 0 | 1 | abs\_addr | |
| безусловный переход  по адресу | PCJumpAddr | JMP loc\_addr | 1 | 0 | 1 | 1 | 0 | 0 | loc\_addr |
| вызов подпрограммы по адресу | LR PC + 2; PCJumpAddr | JAL loc\_addr | 1 | 0 | 1 | 1 | 0 | 1 | loc\_addr |

**Схема организации памяти**



# Описание модулей периферии

Для того, чтобы отличать инструкции от сигналов блоку периферии, введем header. Он будет отправляться первым, и в зависимости от его значения будет либо передана инструкция (значит следующие 16 бит - код инструкции), либо блок периферии будет записывать в регистр какое-то значение (после него сразу идет следующий header). Регистры ctrl0 и ctrl1 служат для управления блоком периферии.

**Формат header'a:**

|  |  |
| --- | --- |
| **Кодировка** | **Действие** |
| 10000000 | Передача инструкции (следующие 16 бит) |
| 01000000 | Запись в status1 - все инструкции загружены |
| 00100000 | Запись в ready 1- разрешение на передачу от платы к ПК |

**Регистр ctrl0**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Биты | Поле | Режим доступа | Описание | Значение по сбросу |
| 31:25 | Reserved | - | - | - |
| 24 | ready | RO | В поле записывается 1 при поступлении header'а - сигнал к началу передачи от платы к ПК | 0 |
| 23 | start | RW | По соответствующему сигналу от памяти устанавливается 1 для приема данных из памяти | 0 |
| 22 | status | RO | В поле пишется 1 по сигналу header'а, когда все данные от ПК к плате переданы | 0 |
| 21:11 | addr\_end | RO | Конечный адрес записи в RAM | Значение не меняется |
| 10:0 | addr\_beg | RO | Начальный адрес записи в RAM | Значение не меняется |

**Регистр ctrl1**

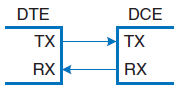
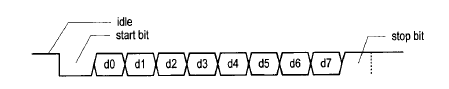
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Биты | Поле | Режим доступа | Описание | Значение по сбросу |
| 31:28 | Reserved | - | - | - |
| 27 | finish | RW | Принимает значение 1, если все параметры от памяти получены | 0 |
| 26:16 | addr | RW | Текущий адрес, полученный от памяти | 0 |
| 15:0 | data | RW | Последние данные, прочитанные из памяти | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Назначение** | **Направление** | **Разрядность** |
| ack\_i | Оповестительный сигнал от ведущего устройства | IN | 1 |
| stb\_i | Индикатор ои ведущего устройства о готовности работы | IN | 1 |
| addr\_i | Входная шина адреса | IN | 16 |
| dat\_i | Входная шина данных | IN | 16 |
| we\_i | Входной сигнал на разрешение записи/чтения | IN | 1 |
| ack\_o | Оповестительный сигнал ведущему устройства | OUT | 1 |
| stb\_o | Индикатор ведущему устройству о готовности работы | OUT | 1 |
| addr\_o | Выходная шина адреса | OUT | 16 |
| dat\_o | Выходная шина данных | OUT | 16 |
| we\_o | Выходной сигнал на разрешение записи/чтения | OUT | 1 |
| tx | Выходная линия последовательной передачи данных | OUT | 1 |
| rx | Входная линия последовательной передачи данных | IN | 1 |
| clk\_i | Тактовый сигнал | IN | 1 |
| rst\_i | Сигнал сброса | IN | 1 |

**UART**

UART (UniversalAsynchronousReceiver-Transmitter) - универсальный асинхронный приемник-передатчик, интерфейс для последовательной передачи данных.

UART состоит из приемника (rx) и передатчика (tx).Передача данных в UART осуществляется по одному биту в равные промежутки времени. Этот временной промежуток определяется заданной скоростью UART и для конкретного соединения указывается в бодах (количество бит в секунду в нашем случае, но это верно только для двоичного кодирования). Помимо собственно информационного потока UART автоматически вставляет в поток синхронизирующие метки, так называемые стартовый и стоповый биты. При приёме эти лишние биты удаляются из потока. В нашем дизайне используется 2 стоповых бита (чтоб затормозил наверняка). Также может посылаться бит четности, но у нас он не используется. Idleline имеет высокий уровень, стартовый бит -низкий, а стоповые биты - тоже высокий.

****

До начала передачи приемник и передатчик должны согласоваться о параметрах передачи - бод рейту, количеству битов данных, количеству стоповых битов, использованию бита четности.

Для того, чтобы приемник мог распознать биты в потоке информации, используется схема выборки с запасом. Обычно частота выборки в 16 раз больше бод рейта - каждый бит сэмплируется 16 раз. В середине каждого бита происходит сохранение его значения. Очевидно, и приемник и передатчик работают на сдвиговых регистрах.

Имеет смысл описать интерфейсные сигналы uart, как время будет

Описать функционал периферии: получении инструкций с ПК, выгрузка данных из памяти и передача к ПК. Как настраивается и запускается данный функционал и т.п.

# Функциональное тестирование и верификация

# Функциональная модель

Симулятор, эмулятор, модель ЭВМ — под этими понятиями подразумевается компьютерная программа, способная имитировать работу некоторой реальной вычислительной системы. Симуляция– процесс, имитирующий внешние проявлениясистемы, внутреннее его устройство при этом не повторяет детально оригинал.

**Функциональная модель** (англ. functional model) — симулятор, точность которого ограничена корректной функциональностью целевых приложений без обеспечения правильных значений длительностей операций, наблюдаемых в реальности. Подобные модели недостаточно точны для предсказания производительности, но, как правило, достаточны для корректной работы.

Модель процессора представляет собой интерпретатор, который работает над входной машинной инструкцией.

Рабочий цикл интерпретатора:

1. Fetch — чтение машинного кода из памяти.
2. Decode — декодирование текущей функции, заключённой в инструкции, а также её аргументов — операндов.
3. Execute — исполнение функции над аргументами.
4. Writeback — запись результатов память.
5. Advance PC — продвижения регистра-указателя инструкций (PC, program counter)

**Реализация модели процессора на СИ**

Переменные, используемые для описания регистров (названия и назначение соответствуют указанным в документации), имеют тип int (32-разрядные). Моделируемая память представляет собой массив данных, адрес которых и значения совпадают для простоты проверки работы модели (1024 элемента). Также есть возможность записывать данные в массив памяти из тестового окружения по конкретным адресам.

Модель процессора имеет два режима работы:

1. Работа с бинарным файлом.
2. Работа с тестовым окружением через DPI- подключение.

Табл.1. Этапы реализации.

|  |  |  |  |
| --- | --- | --- | --- |
| Название функции | Аргументы | Возвращаемое значение | Назначение |
| init\_memory |  | memory[addr] | Инициализация массива памяти |
| open\_file |  | -1 – в случае ошибки  0 – выполнено | Открытие бинарного файла с функцией его выбора |
| read\_instr |  | instr | Чтение инструкции из файла, управляемое изменением программного счетчика PC. Считывается по 8 бит информации за 1 цикл. Инструкция сохраняется в буфере instr. |
| decode | int instr,  int verbosity | Зависит от исполняемой операции | Декодирование инструкции. Определяются opcode операции и, в зависимости от него, аргументы rs, rt, rd, imm, addr. Далее, с помощью оператора switch, осуществляется вызов и исполнение заданной операции. В зависимости от операции производится запись результата по указанному адресу в регистр или в память. |
| allreg | int verbosity |  | Вывод на консоль значений регистров. |
| statistics | int verbosity |  | Вывод на консоль информации о количестве и типе выполненных инструкций. |
| write\_results |  |  | Запись результатов в файл. |

Примечание. Аргумент verbosity необходим для включения(verbosity = 0) и отключения(verbosity = 1) консольного вывода. В данном режиме для удобства использования verbosity = 0. Но в случае обработки большого количества инструкции удобнее использовать только запись в файл, при этом промежуточные вычисления записаны не будут (только результаты и статистика).

**Взаимодействие с тестовым окружением.**

Для взаимодействия модели и тестового окружения используется такой интерфейс, как DPI, который обеспечивает простой и эффективный способ подключения кода на каком-либо языке программирования к SystemVerilog. В данном случае идет подключение кода на C.

Функции, реализованные в C и указанные в импортных объявлениях, могут быть вызваны в SV и называются импортируемыми.

Функции, реализованные в SV и указанные в экспортных объявлениях, могут быть вызваны в C и называются экспортируемыми.

Пример объявления импортируемой функции в С:

DPI\_LINK\_DECL DPI\_DLLESPEC decode(int, int);

В данном режиме реаботы можно как проверять отдельно каждую операцию, так и посылать инструкции в функцию decode. Функции, которые могут быть импортированы в TB, представлены в Таблице 2.

Таблица 2.Функции, используемые тестовым окружением

|  |  |  |
| --- | --- | --- |
| Название функции | Аргументы | Назначение |
| int ADD(int, int, int) | rs, rt, rd | Сложение |
| int ADDI (int, int, int) | rs, rt, rd | Сложение с константой |
| int OR (int, int, int) | rs, rt, rd | Побитовое ИЛИ |
| int AND (int, int, int) | rs, rt, rd | Побитовое И |
| int XOR (int, int, int) | rs, rt, rd | Побитовое исключающее ИЛИ |
| int NOR (int, int, int) | rs, rt, rd | Побитовое отрицание |
| int SLL (int, int, int) | rs, rt, rd | Логический сдвиг влево |
| int ROT (int, int, int) | rs, rt, rd | Циклический сдвиг влево |
| int BNE (int, int, int) | rs, rt, rd | Условное ветвление, если не равны |
| int LDL (int) | addr | Загрузка из памяти в регистр |
| int LDH (int) | addr | Загрузка из памяти в регистр |
| int STL (int) | addr | Сохранение из регистра в память |
| int STH (int) | addr | Сохранение из регистра в память |
| int JMP (int) | addr | Безусловный переход по адресу |
| int JAL (int) | addr | Вызов подпрограммы по адресу |
| int JR (int) | addr\_rs | Безусловны переход по значению регистра |
| int JALR (int) | addr\_rs | Вызов подпрограммы по значению регистра |
| NOP() |  | Нет операции |
| decode(int, int) | instr, verbosity | Декодирование инструкции с последующим исполнением |
| int init\_memory() |  | Вызов заранее заданного в программе массива памяти |
| set\_memory(int, int) | addr, data | Загрузка массива памяти из TB |
| int get\_memory(int) | addr | Запрос данных по указанному адресу в памяти |
| int SETREG(int, int) | addr\_reg, data\_reg | Загрузка данных по адресу регистра из TB |
| int GETREG(int) | addr\_reg | Запрос данных по указанному адресу регистра |
| allreg(int) | verbosity | Вывод на консоль данных всех регистров, если verbosity = 0, отключение вывода, если verbosity = 1 |
| statistics(int) | verbosity | Вывод на консоль данных о выполненных инструкциях, если verbosity = 0, отключение вывода, если verbosity = 1 |
| write\_results() |  | Вывод результатов в текстовый файл |

**Алгоритм построения цифрового дайджеста MD4**

Предполагается, что на вход подано сообщение, состоящее из ~b~ бит.

### Шаг 1. Добавление недостающих битов.

Расширение производится следующим образом: один бит, равный 1, добавляется к сообщению, а затем добавляются биты, равные 0, до тех пор, пока длина сообщения не станет равной 448 по модулю 512. Таким образом, в результате расширения, сообщению недостает 64 бита до длины, кратной 512 битам. Расширение производится всегда, даже если сообщение изначально имеет нужную длину.

### Шаг 2. Добавление длины сообщения.

64-битное представление ~b~ добавляется к результату предыдущего шага. На этом этапе мы получаем сообщение длиной кратной 512 битам. Это эквивалентно тому, что это сообщение имеет длину, кратную 16-ти 32-битным словам. Каждое 32-битное слово содержит четыре 8-битных, но следуют они не подряд, а наоборот. Пусть M[0 \ldots N-1] означает массив слов получившегося сообщения (здесь N кратно 16).

В данной работе было произведено построение цифрового дайджеста «abc».

Расширение сообщения не было программно реализовано в связи с большой сложностью (Шаг 1 и Шаг2), поэтому его необходимо производить вручную. Дальнейшие шаги описаны на языке ассемблера для заданной архитектуры.

### Шаг 3. Инициализация MD-буфера.

Для вычисления хеша сообщения используется буфер, состоящий из 4 слов (32-битных регистров): ~(A,B,C,D).

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

### Шаг 4. Обработка сообщения блоками по 16 слов.

Вспомогательные функции, каждая из которых получает на вход три 32-битных слова, и по ним вычисляет одно 32-битное слово:

F(X,Y,Z) = XY \lor \neg X Z

G(X,Y,Z) = XY \lor XZ \lor YZ

H(X,Y,Z) = X \oplus Y \oplus Z

Алгоритм хеширования на абстрактном языке программирования для сообщения «abc»:

Загружаем в память 16 блоков 32-разрядных слов

Загружаем в память необходимые константы

Сохраняем A как AA, B как BB, C как CC, и D как DD

AA = A

BB = B

CC = C

DD = D

Раунд 1

Пусть [abcd k s] означает следующую операцию:

a = (a + F(b,c,d) + X[k]) <<< s

[ABCD 0 3] [DABC 1 7] [CDAB 2 11] [BCDA 3 19]

[ABCD 4 3] [DABC 5 7] [CDAB 6 11] [BCDA 7 19]

[ABCD 8 3] [DABC 9 7] [CDAB 10 11] [BCDA 11 19]

[ABCD 12 3] [DABC 13 7] [CDAB 14 11] [BCDA 15 19]

Раунд 2

Пусть [abcd k s] означает следующую операцию:

a = (a + G(b,c,d) + X[k] + 5A827999) <<< s

[ABCD 0 3] [DABC 4 5] [CDAB 8 9] [BCDA 12 13]

[ABCD 1 3] [DABC 5 5] [CDAB 9 9] [BCDA 13 13]

[ABCD 2 3] [DABC 6 5] [CDAB 10 9] [BCDA 14 13]

[ABCD 3 3] [DABC 7 5] [CDAB 11 9] [BCDA 15 13]

Раунд 3

Пусть [abcd k s] означает следующую операцию:

a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s

[ABCD 0 3] [DABC 8 9] [CDAB 4 11] [BCDA 12 15]

[ABCD 2 3] [DABC 10 9] [CDAB 6 11] [BCDA 14 15]

[ABCD 1 3] [DABC 9 9] [CDAB 5 11] [BCDA 13 15]

[ABCD 3 3] [DABC 11 9] [CDAB 7 11] [BCDA 15 15]

Затем производим следующие операции сложения.

A = A + AA

B = B + BB

C = C + CC

D = D + DD

### Шаг 5. Формирование хеша.

Цифровой дайджест сообщения представляет собой результат вывода содержимого регистров A, B, C, D. То есть мы начинаем с младшего байта A и заканчиваем старшим байтом D. Это завершает описание алгоритма MD4.

# Описание тестового окружения

# Прототипирование