УДК 004.27 DOI: 10.31114/2078-7707-2020-3-158-164

Архитектура процессоров большой разрядности: проблемы и решения

В.В. Ерохин

АО «НИИМА» ПРОГРЕСС», г. Москва, vladimir.v.erokhin@gmail.com

***Аннотация* — Описаны проблемы, возникающие при проектировании перспективных процессорных архитектур большой разрядности. Рассмотрены принципы организации таких процессоров, на этой основе предложены архитектура процессоров большой разрядности и применение этих архитектур.**

***Ключевые слова* — архитектура, процессор, система команд.**

I. Введение

Несмотря на то, что в настоящее время существует большое количество различных процессоров, в мире постоянно появляются новые архитектуры, постоянно идет поиск новых решений. Одной из наиболее важных причин этого является требование увеличения точности, а значит, разрядности регистров процессора. Примеры привести очень легко. Когда-то считались достаточными процессоры с 16-разрядным словом, далее появились 32-битые процессоры, но и они с определенного момента стали уступать место 64-битным. А в последнее время появились и 128-битные архитектуры. Можно предполагать, что в недалеком будущем возникнет потребность в процессорах еще большей разрядности. Как построить систему команд таких процессоров? Какие принципы следует положить в основу построения этой системы команд? Как не потерять при этом возможности «старых» процессоров в обработке данных небольшой разрядности?

Данная работа — это попытка ответить на перечисленные вопросы.

В табл. 1 приведены используемые в работе обозначения.

II. Архитектура процессоров RISC**2 (R^2T)

A. Требования к архитектуре R^2T

Развитие современных процессоров идет по нескольким путям.

Фирмы-монополисты, владеющие правами на архитектуру процессоров определенных типов, как правило, развивают свои архитектуры «снизу-вверх». Изначально 32-битная архитектура ARM, MIPS, и, будем считать, Intel, постепенно преобразовывалась в 64 битную (миграции в меньшую разрядность менее характерны, потому рассматривать их нет смысла). Это объясняется ростом реальной потребности в точности вычислений, и, что не последнее в этом деле, истечением сроков патентной защиты архитектур. Фирмы – владельцы патентов расширяют разрядность и одновременно архитектуру своих процессоров и вновь защищают их патентами. Делается это далеко не всегда обоснованно с точки зрения необходимости таких изменений для реального развития архитектуры и тем более уж никак для удобства пользователя. В результате, когда-то вполне осмысленные архитектуры разрастаются настолько, что именовать их RISC становится явной натяжкой.

Второй подход заключается в развитии открытых архитектур, среди которых наиболее известны OpenRISC и RISC-V, в которых нет необходимости безосновательно наращивать число команд. Но и эти архитектуры не могут быть застывшими, поскольку требования к процессорам все время меняются.

В частности, процессор RISC-V расширяет разрядность процессора до 128 при относительно небольшом числе операционных кодов. Но рано или поздно и эти показатели станут недостаточными.

Кроме увеличения разрядности, можно указать несколько тенденций в развитии, которые, в частности, отмечены в описании RISC-V. [1]

Это обеспечение большой адресуемой памяти, которая в дальнейшем может потребоваться в реальности, и наконец, сокращение числа команд процессора за счет удаления излишних, то есть, по существу, возврате к принципам RISC, а также реализация архитектур SIMD, VLIW в рамках расширения архитектуры.

Что касается длины командного слова, можно отметить, что максимальная длина машинного слова в известных процессорах не превышает 32, исключая специализированные архитектуры VLIW.

В целом для практически неограниченного развития архитектуры можно сформулировать требования, которым она должна удовлетворять:

- архитектура не должна задавать ограничения на дальнейшее увеличение разрядности;
- адресуемая память должна иметь возможность расти с увеличением разрядности процессора;
- число команд не должно расти с увеличением разрядности процессора;

• длину командного слова сверх обычной (32) желательно не увеличивать за исключением архитектур типа VLIW.

Итак, для разработки основ архитектуры перспективных процессоров R^2T надо решить следующие задачи:

- обеспечить неограниченную разрядность процессора;

- обеспечить неограниченное адресное пространство;

- разработать систему команд, не зависящую от разрядности процессора и этим обеспечить совместимое ПО для процессоров разной разрядности снизу-вверх;

Таблица 1

Используемые обозначения

w8, w16, w32, w64, w128, w256, w512, w1024, w2048…	Размеры операнда, соответственно: байт, 16-битное слово, 32-битное слово, 64-битное слово, 128-битное слово, 256-битное слово, 512-битное слово, 1024-битное слово, 2048-битное слово.
qualifier	Параметр команды, определяющий размер операнда, к которому относится операция. Допустимые значения {**w8, w16, w32, w64, w128, w256, w512, w1024, w2048, etc.**}
offset**xx**	Смещение адреса в виде знаковой константы длиной **xx** бит, содержащейся непосредственно в командах
imm**xx**	Непосредственные данные со знаком или без знака (определяется командой), содержащиеся в коде команды длиной **xx** бит.
<<	Сдвиг влево на указанное далее число разрядов
==	Символ эквивалентности
c(xlen-1)	бит переноса из разряда (xlen-1)
succ	Следующий элемент, в данном тексте подразумевается следующий по номеру регистр

B. Основные принципы построения архитектуры R^2T

Одна из главных проблем при переходе к процессору большей разрядности – это работа с константами, которые используются в операциях с непосредственными операндами. Если процессор имеет 32-битные команды, то максимальная длина константы составляет, как правило, не более 24 бит. Это может быть приемлемым для 32 битных, менее приемлемо для 64 битных процессоров, но совсем плохо работает для процессоров с длиной операндов, большей 64 бит.

Сформулируем **Принцип 1**.

Константы можно только загружать, в том числе и непосредственно (за исключением небольшого числа операций).

Еще одна проблема в традиционных архитектурах – выделение операндов некоторых разрядностей, как правило, равных степени двойки, при выполнении различных операций, таких как сложение, вычитание, умножение, деление. Например, сложение слова, сложение двойного слова и т. д. При увеличении разрядности процессора число таких операндов, для которых существуют отдельные команды, растет, особенно учитывая операции с непосредственными данными.

Отсюда **Принцип 2**.

Отказ от понятия слова в плане выделения данных некоторой размерности в операциях.

То есть, операции с байтом, двумя байтами, четырьмя байтами… должны быть полностью равноценны в смысле отсутствия специальных команд для работы с отдельными размерами данных в одинаковых операциях, вроде (add, dadd), …

И наконец, **Принцип 3.**

Набор инструкций (базовый и расширения системы команд) одинаков для всех вариантов процессоров от 32 (16 или даже 8) битных до максимума и, желательно, всех расширений архитектуры.

Приведенные принципы позволяют описать требования, предъявляемые к архитектуре процессоров R^2T:

а) Количество регистров общего назначения – 32.

Обычно вопрос о числе регистров обходят при описании процессоров, выбор остается без пояснений.

Однако, имеются исследования, которые показывают, что 32 регистра вполне достаточно для подавляющего числа приложений, а 16 регистров не очень сильно ограничивают возможности процессора, оценивая разницу в производительности процессора с 16 и 32 регистрами в 5 процентов. [2]

Исходя из этого число регистров в процессоре выбрано 32, а для процессоров встраиваемых предложений – 16.

b) Разрядность регистров может варьироваться от 32 (реально 16 или даже 8) до 2048 и выше.

c) Команды располагаются в памяти строго выравненными на 32-разрядное слово. Передача управления выполняется строго на границу 32-разрядного слова. Значения смещения в командах перехода трактуется применительно к 32-разрядному слову. Сказанное относится к 32-битным командам. В командах 16-битных все выравнивания осуществляется на границу двух байтов (четные адреса).

Последнее правило сильно упрощает выборку команд.

Теперь можно перейти к проектированию системы команд процессорной архитектуры R^2T.

Сложно сказать что-либо о принципах выбора базового и дополнительных наборов команд, которыми руководствовались создатели RV, но следует обозначить эти принципы достаточно четко.

Во-первых, каждый, в том числе и базовый набор должен легко реализовываться на одном компактном устройстве.

Для выполнения операций умножения требуется аппаратный умножитель, если, разумеется, не выполнять его микропрограммным способом. Умножитель – это отдельное довольно объемное устройство. Умножение может выполняться за один или за несколько тактов, в пределе за N тактов, где N – разрядность процессора.

Команда деления тоже требует отдельного и тоже довольно большого устройства, которое никак не может разделять аппаратуру с другими устройствами. Команда деления также может выполняться последовательно за N тактов, если результат получается один бит за такт или быстрее, если за один такт будет получено несколько бит. В принципе, можно получить результат и за один такт, но это будет довольно медленно с точки зрения длительности такта и затратно при N > 16. [3]

То же самое можно сказать и о команде подсчета лидирующих единиц, с той оговоркой, что по сложности устройство подсчета лидирующих единиц много проще умножителя и делителя.

Дополнительным критерием для выбора расширенных наборов команд является возможность разделения аппаратных ресурсов при реконфигурации процессоров.

C. Формат команд процессоров для встраиваемых приложений

К процессорам для встраиваемых приложений по сравнению с «обычными» процессорами предъявляют особые требования.

Во-первых, он должен быть максимально компактным, поскольку площадь может быть одним из важнейших факторов.

Во-вторых, как правило, такие процессоры используются в контроллерах, в которых может очень важным время реакции на внешние события, то есть он должен быть процессором «реального времени».

Поэтому для встраиваемых приложений разработана компактная 16-битная система команд, позволяющая обойтись минимальным использованием обычных 32-битных команд для разработки программ.

Итак, можно описать основные особенности архитектуры встраиваемых процессоров.

Процессор для встраиваемых приложений имеет:

- 16 регистров;
- разрядность регистров – от 8 до 64 бит;
- несколько наборов регистров для обеспечения работы с прерываниями в реальном масштабе времени (от 1 до 16 наборов);
- система команд используется в основном редуцированная, то есть 16-битовая, могут быть использованы любые команды из основного набора.
- команды из сокращенного набора выравниваются на 2 байта, если используются дополнительные команды из основного набора, то они выравниваются на 4 байта;
- адреса назначения в командах перехода из сокращенного набора команд выравниваются на 2 байта, в командах из основного набора, если они используются, - на 4 байта.

В принципе, достаточно из основного набора добавить всего лишь несколько команд для нормальной работы процессора встраиваемых приложений, однако, процессор должен выполнять все команды основного набора. При этом в последних поле регистров сокращается с 5 до 4 битов, старший бит игнорируется.

Еще раз отметим, что обычный процессор должен выполнять все команды набора для встраиваемых приложений.

D. Операции процессоров R^2T

Приведем операции, принятые в архитектуре R^2T:

1) *арифметические*

add rd, rs1, rs2 –> rd = rs1 + rs2

sub rd, rs1, rs2 –> rd = rs1 - rs2

sext qualifier, rd, rs – расширение знака операнда. rd = sign_extend(rs).

2) *Контроль арифметических операций add и sub.*

oflchck qualifier – осуществляет проверку переполнения сразу после арифметических операций сложения и вычитания. В случае переполнения выполняется исключение по переполнению. В качестве адреса возврата сохраняется значение IP в регистре 1F (0F для встраиваемых предложений).

Очевидно, такой подход для регистрации переполнения требует сохранения вектора переносов. Он устанавливается только после операций сложения и вычитания, в других ситуациях является неопределенным. Этот вектор недоступен для программиста.

Переполнение в операциях на байтах, например, будет в случае, если c(6) xor c(7) = 1. Здесь c(6) – перенос из 6-го разряда, а c(7) – перенос из 7-го разряда в результате суммирования.

Такой подход позволяет выполнять операции не только над продекларированными размерностями данных (слово, двойное слово, четверное слово), как это сделано в RV, но и над операндами любой размерности, в том числе типа w8, w16. При увеличении размера слова (в эпоху i8086-80286 размер слова считался равным 16, далее он стал равен 32, однако в связи с дрейфом в сторону больших разрядностей не исключено, что слово будет длиной 64 бита, а то и 128) команды не меняются.

3) логические

and rd, rs1, rs2 –> rd = rs1 & rs2

or rd, rs1, rs2 –> rd = rs1 | rs2

xor rd, rs1, rs2 –> rd = rs1 ^ rs2

not rd, rs1 –> rd = !rs1

shl rd, rs1, rs2 –> rd = rs1 <- rs2 сдвиг логический влево на (rs2) разрядов

shr rd, rs1, rs2 -> rd = rs1 -> rs2 сдвиг логический вправо на (rs2) разрядов

sar rd, rs1, rs2 –> rd = rs1 -> rs2 сдвиг арифметический вправо на (rs2) разрядов

4) загрузка

mvs rd, imm16 – Записать непосредственные данные в R1 со сдвигом влево (вдвинуть 16 бит слева)

5) передачи управления

a) безусловные переходы:

auipc rd, offset20 - операция загрузки в rd значения pc + (offset20<<12).

jal rd, offset20 - переход по адресу pc + offset20, адрес возврата записать в rd

jalr rd, rs, offset11 - переход по адресу rs+ offset11, адрес возврата записать в rd

b) условные переходы:

bz rs, offset16 - если rs = 0, перейти по адресу pc + offset16

bg rs, offset16 - если rs > 0, перейти по адресу pc + offset16. rs – целое со знаком.

bgu rs, offset16 - если rs > 0, перейти по адресу pc + offset16. rs – целое без знака.

6) операции чтения/записи

ld qualifier, rd, rs, offset9 – читать данные в rd по адресу rs с расширением знака.

st qualifier, rd, rs, offset9 – записать данные из rs по адресу rd.

Адреса должны быть выравнены: адрес **w16** имеет 0 в младшем разряде, адрес **w32** – два нуля, **w64** – три и т. д. В случае нарушения этого требования генерируется исключение по адресу.

Все команды могут использоваться в SIMD и векторных процессорах.

7) Операции чтения/записи в регистры управления/статуса процессора

csrwr qualifier, rd, rs

csrrd qualifier, rd, rs

В процессоре предполагается наличие пользовательского регистрового файла регистров управления/статуса CSR. В связи с тем, что разрядность процессора может варьироваться от 8 и выше, а разрядность регистров CSR целесообразно ограничить снизу 32 битами, необходимо организовать побайтовый доступ из, допустим, 8-разрядного процессора к 64-битному CSR. Для этого необходим параметр **qualifier**, который определяет место записи в регистре с адресом rd || 000000b. То есть, в этом случае каждому регистру CSR присваивается диапазон адресов от rd || 000000b до rd || 111111b.

Команда csrwr осуществляет запись из регистра rs в регистр CSR rd.

Команда csrrd осуществляет чтение из регистра CSR rd в регистр rs.

8) системные

ecall – вызов операционной системы, например, при окончании работы программы.

ebreak – Инструкция ebreak используется отладчиками, чтобы вернуть управление назад в среду отладки.

Примечание. Данные команды имеют длину 16 бит. При использовании с 32-битным основным набором команд следует использовать после любой из них 16-разрядную команду **nop** (**add** r0, r0, r0).

E. Расширения системы команд процессоров R^2T

1) Расширение 0. Операции с непосредственными данными.

Как было декларировано в самом начале, в предлагаемом варианте системы команд применение команд с непосредственными операндами ограничено. Однако, чтобы не доводить до абсурда, в случаях, когда противоречия с будущими приложениями системы команд не предвидятся, можно использовать опционально данное расширение.

В основном, описанные ниже команды вынесены в расширение системы команд для того, чтобы подчеркнуть особенность процессора R^2T.

addi rd, rs1, imm11 –> rd = rs1 + imm11

Команда сложения с непосредственным операндом не приведет к каким-либо проблемам в обозримом увеличении разрядности процессоров. Непосредственный операнд используется, как правило, в качестве шага

цикла и т. п. случаев, и конечно, для небольших арифметических операндов.

shli rd, rs1, imm11 –> rd = rs1 <- rs2 сдвиг логический влево на (imm11) разрядов

shri rd, rs1, imm11 -> rd = rs1 -> rs2 сдвиг логический вправо на (imm11) разрядов

sari rd, rs1, imm11 –> rd = rs1 -> rs2 сдвиг арифметический вправо на (imm11) разрядов

Команды сдвигов могут стать неактуальными лишь при возрастании разрядности регистров выше 2048, что опять-таки не приведет к каким-либо проблемам в ближайшем будущем, поскольку целесообразность процессоров с более высокой разрядностью пока не просматривается.

2) Расширение S. Операции загрузки регистрового набора в стек и выгрузки из стека

pusha

popa

В процессоре допускается наличие аппаратного стека, который может быть использован при вызове прерываний и обычных процедур. Элементами стека являются регистровые наборы. При выполнении команды pusha содержимое текущего набора выгружается в аппаратный стек. Если при выполнении команды pusha аппаратный стек заполнен, формируется исключение по записи в стек, при обработке которого последний элемент стека должен выгружаться в память.

При выполнении команды **popa** данные перезаписываются из стека в текущий набор. Если состояние стека таково, что еще остаются данные в памяти, то формируется исключение по чтению из стека, при обработке которого данные из памяти перезаписываются в дополнительный элемент стека. После возврата из исключения регистровый файл восстанавливается из стека, причем в последний элемент аппаратного стека записываются данные из дополнительного элемента стека.

Команды работы со стеком выполняются за один такт, если не происходит исключения.

Если аппаратный стек отсутствует, но команды **pusha**, **popa** используются, то исключения по переполнению аппаратного стека и по чтению из стека выполнятся всегда при выполнении этих команд.

Примечание. Данные команды имеют длину 16 бит. При использовании с 32-битным основным набором команд следует использовать после любой из них 16-разрядную команду **nop** (**add** r0, r0, r0).

3) Расширение M. Операции умножения

muls rd, rs1, rs2 –> (rd, succ(rd)) = rs1* rs2**,** знаковое умножение

mulu rd, rs1, rs2 –> (rd, succ rd)) = rs1* rs2, беззнаковое умножение

Некоторые необходимые пояснения.

Принятый в RV порядок умножения представляется чересчур жестким. Выполнять команду умножения дважды для получения полного результата, как это принято в RV, – это явный перебор, поддержка исходных постулатов за счет довольно большого увеличения потребляемой мощности и времени выполнения не является обоснованной. При том, что при операндах большой разрядности эта операция не будет выполняться за один такт, поскольку это приведет к большому снижению тактовой частоты, вполне можно дополнительно потратить еще один такт к собственно умножению, и дописать младшую половину результата в нужный регистр.

Второй вопрос, что следует использовать в качестве второго регистра результата.

Универсальным вариантом было бы иметь возможность в качестве такого регистра любой регистр, кроме того, куда будет направлена старшая часть результата. Тогда команда могла бы выглядеть так:

mul (rd1, rd2), rs1, rs2, где пара **(rd1, rd2)** предназначена для результата умножения.

Однако, не сильно уменьшая общность, проще в реализации будет

mul rd, rs1, rs2, где пара **(rd,** succ**(rd))** предназначена для результата умножения. То есть, результат запоминается в последовательной паре регистров.

Можно было ввести в систему команд команды умножения с суммированием (madd, msub), но в общем случае эти команды будут выполняться за несколько тактов, чтобы избежать снижения тактовой частоты. Но тогда более эффективно будет разбить команду умножения с суммированием на последовательное выполнение команды умножения и суммирования. Поскольку, как сказано выше, по результату сложения должен храниться вектор переносов, то суммирование двух пар регистров, где хранятся текущая сумма и результаты последней команды умножения, не представляет сложности. Таким образом, достаточно ввести команды

addc rd, rs1, rs2 Сложение с переносом rd = rs1 + rs2 + c(xlen-1)

subb rd, rs1, rs2 Вычитание с заемом rd = rs1 - rs2 – c(xlen-1)

Умножение с суммированием будет выглядеть так. Допустим, текущая сумма хранится в паре (r5, r6), операнды находятся в регистрах (r10, r11), а результат умножения будет получен в паре (r7, r8). Тогда тройка операций

mul r7, r10, r11

add r6, r6, r8

addc r5, r5, r7

реализует операцию умножения с суммированием (r5, r6) = (r5, r6) + r10 * r11 с использованием регистров r7, r8.

4) Расширение D. Операции деления

divs rd, rs1, rs2 –> (rd, succ**(rd)) = rs1/ rs2,** знаковое деление

divu rd, rs1, rs2 –> (rd, succ**(rd)) = rs1/ rs2,** беззнаковое деление

Результат, как и в умножении, частное и остаток запоминаются в последовательной паре регистров.

Деление вынесено в отдельное от умножения расширение, так как умножение требуется значительно чаще, чем деление, и выполняется значительно быстрее.

5) *Расширение B. Операция над битовыми полями*
clo rd, rs – подсчет числа лидирующих единиц. Источник – регистр rs, число единиц – в rd.

6) *Расширение A. Атомарное чтение и запись.*
ll qualifier, rd, rs, offset11

sc qualifier, rd, rs, offset11

Атомарное чтение данных с расширением знака в регистр rd из памяти по адресу rs + offset12, где rs – содержимое регистра rs (база), offset12 – смещение без знака.

Атомарная запись данных из регистра rd в память по адресу rs + offset12, где rs – содержимое регистра rs (база), offset12 – смещение без знака. Если запись успешна, в регистр rd записывается 1 и 0 в противном случае.

Параметр **qualifier** определяет тип загружаемых данных и выравнивание адреса. Если **qualifier** = **w8**, то адрес может быть произвольным, если **w16** – кратен 2, **w32** – 4, **w64** – 8 и т. д.

F. Регистровый файл процессоров R²T в базовом исполнении

Регистровый файл R²T в базовом исполнении состоит из 32 регистров общего назначения, структура его приведена на рис.1. Параметр xLEN может принимать значения от 8 до 2048 и выше.

xLEN-1 0

x0(==0)
x1
x2
x3
x4
x5
x6
x7
x8
x9
x0A
x0B
x0C
x0D
x0E
x0F
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x1A
x1B
x1C
x1D
x1E
x1F

PC

Рис. 1. Регистровый файл базового набора команд

G. Регистровый файл процессоров R²T для встраиваемых приложений

Для встраиваемых приложений, как было сказано выше, количество регистров уменьшено до 16, структура его приведена на рис.2. Параметр xLEN здесь может принимать значения от 8 до 64.

xLEN -1 0

x0(==0)
x1
x2
x3
x4
x5
x6
x7
x8
x9
x0A
x0B
x0C
x0D
x0E
x0F

PC

Рис. 2. Регистровый файл набора команд для встраиваемых приложений

H. Регистры управления/состояния

На рис. 3 изображен рисунок регистра управления/состояния. Параметр csrLEN может принимать значения от 32 и до xLEN в базовой архитектуре и 32 в варианте для встраиваемых приложений.

csrLEN -1 0

CSR

Примечание 1. Адрес записи или чтения ADDR = rd * (csrLEN/8), если xLEN < 32, иначе rd.

Примечание 2. csrLEN = xLEN, если xLEN > 32, иначе 32

Рис. 3. Регистр CSR

I. Полное описание системы команд процессоров R^{2}T

Полное описание системы команд будет дано в другой работе.

J. Исключения в процессорах R^{2}T

Определены следующие исключения:

- аппаратный сброс (reset)
- исключение по переполнению;
- исключение по адресу записи/чтения;
- исключение по записи в стек;
- исключение по чтению из стека;
- исключение по сигналу внешнего прерывания.

Возможно введение дополнительных исключений.

Векторные прерывания организуются при помощи внешнего контроллера прерываний, на который могут поступать сигналы от внутрипроцессорных источников прерываний (счетчик тактов, таймеры, программные прерывания…). От контроллера в процессор поступает номер обрабатываемого прерывания, по которому вычисляется адрес процедуры прерывания (вектор).

K. Порядок байтов в процессорах R^{2}T

В архитектуре процессоров R^{2}T подразумевается использование порядка байтов Little-Endian.

III. ВЫВОДЫ

1. Рассмотрены основные проблемы, возникающие при проектировании процессоров большой размерности, и предложены принципы разработки архитектур процессоров большой размерности.

2. Предложена система команд для построения процессоров высокой размерности, независимая от длины операндов длиной от 8 до 2048 бит и более и включающая наборы команд для встраиваемых процессоров.

3. Поставлены задачи уточнения системы команд и реализации процессорных структур на базе процессоров высокой разрядности.

В заключение отметим, что несмотря на довольно большой опыт в разработке процессоров, автор вполне сознает его ограниченность и предлагает сообществу развивать и уточнять систему команд и конфигурации процессоров R^{2}T.

Литература

[1] The RISC-V Instruction Set Manual. Volume I: User-Level ISA. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf. [Electronic resource]. – Mode of access: https://riscv.org. – Date of access: 24.03.2020.

[2] Diamond Standard Processor Cores // ip.cadence.com [Electronic resource]. – Mode of access: https://ip.cadence.com. – Date of access: 24.03.2020.

[3] Erokhin, V.V. Single-clock division algorithm / V.V. Erokhin // OpenCores [Electronic resource]. – Mode of access: http://opencores.com. – Date of access: 24.03.2020.

Multi-bit Processors Architectures: Problems and Solutions

V.V. Erokhin

NIIMA PROGRESS JSC, Moscow, vladimir.v.erokhin@gmail.com

***Abstract* - Multi-bit processor architecture design problems are described. The processors organization principles are considered, the multi-bit processor architecture applications are proposed.**

***Keywords* - architecture, processor, instruction system.**

References

[1] The RISC-V Instruction Set Manual. Volume I: User-Level ISA. https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf. [Electronic resource]. – Mode of access: https://riscv.org. – Date of access: 24.03.2020.

[2] Diamond Standard Processor Cores // ip.cadence.com [Electronic resource]. – Mode of access: https://ip.cadence.com. – Date of access: 24.03.2020.

[3] Erokhin, V.V. Single-clock division algorithm / V.V. Erokhin // OpenCores [Electronic resource]. – Mode of access: http://opencores.com. – Date of access: 24.03.2020.