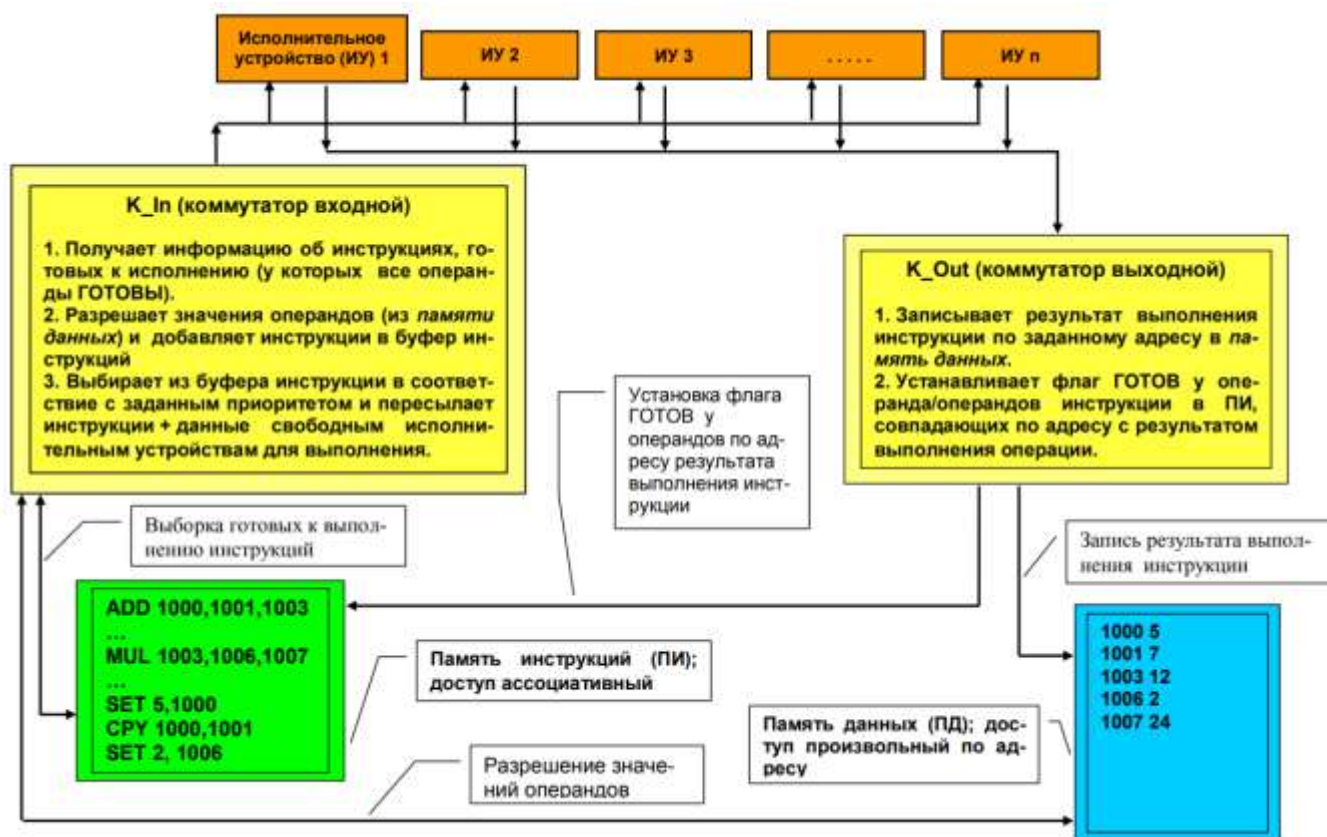


ОСНОВНЫЕ ПОЛОЖЕНИЯ ПРОГРАММНОГО СИМУЛЯТОРА Data-Flow ВЫЧИСЛИТЕЛЯ СТАТИЧЕСКОЙ АРХИТЕКТУРЫ С АППАРАТНЫМ МАССОВЫМ РАСПАРАЛЛЕЛИВАНИЕМ ОПЕРАЦИЙ НА УРОВНЕ ИНСТРУКЦИЙ ПРОЦЕССОРА С ПРЕДИКАТНЫМ ВЫПОЛНЕНИЕМ АРИФМЕТИКО-ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Выгрузить инсталляционную версию (формат Win'32, GUI) данной программы можно здесь: http://vbakanov.ru/dataflow/content/install_df.exe ; дополнительная информация о проекте тут: <http://vbakanov.ru/dataflow/dataflow.htm>

Т.к. программный модуль **data_flow.exe** при работе использует некоторые данные с сервера разработчика, будьте готовы при первом запуске программы после сообщения брандмауэра Защитника Windows разрешить доступ указанного исполняемого файла к серверу разработчика (впрочем, запрет доступа существенно не ухудшит качество работы программы).



Далее везде будем использовать термин *оператор* вместе (или вместо) более точного термина *машинная инструкция* (или просто *инструкция*) в основном в целях сокращения написания и упрощения понятия.

1. **Вычислитель** включает отдельные банки памяти для инструкций (ПИ) и данных (ПД) и множество однотипных (а возможно, и специализированных) процессоров с равноприоритетной схемой доступа к общей памяти данных (**SMP** - *Symmetric Multi-Processors*). **Напомним, что в классической схеме DATA-FLOW вычислителя наличествует общая ассоциативная память для инструкций и данных (данные хранятся вместе с инструкциями и тегами);** данное изменение сделано для упрощения понимания работы вычислителя и соответствует принципу *симулирования* объекта (моделируются только наиболее важные внешние его свойства без конкретизации особенностей функционирования).

Т.к. эти процессоры, вообще говоря, упрощены по сравнению с традиционными, их относят к одной из групп *исполнительных устройств* (ИУ) и часто называют арифметическими *исполнительными устройствами* (АИУ); они, конечно, могут быть самыми различными - напр., векторными АИУ, устройствами ввода-вывода и др.

Данный симулятор предназначен в основном для изучения *динамики выполнения программ* (учитывается только время выполнения инструкций, время доступа к памяти считается нулевым) на потоковом вычислителе, поддерживает алгоритмы с условным и безусловным выполнением отдельных частей (методом *предикатного выполнения*) и требует следования концепции единократного присваивания (*).

2. Доступ к ПД адресный, роль адреса выполняет имя (*уникальный идентификатор*) переменной; доступ к ПИ – ассоциативный (*в физической реализации, в данном симуляторе моделируется*); *порядок следования инструкций в DATA-FLOW вычислителе несущественен* (в симулятор специально введена возможность случайного перемешивания инструкций с последующим выполнением).
3. Имена операндов и результат выполнения инструкций суть произвольные 32-символьные последовательности (без пробелов, запятых и символа прямого слэша). Каждая вновь введённая переменная создаётся, когда соответствующее имя появляется в *поле результата инструкции* и в дальнейшем может служить только входным операндом в иных инструкциях; в случае попытки изменения значения выдаётся предупреждение, позволяющее перезаписать значение или отказаться от этого.
4. Формально все имена операндов (*входных переменных*) каждой инструкции должны хотя бы раз встретиться в качестве *результата иной инструкции*; до этого момента они считаются неопределёнными; перед началом выполнения программы проводится поиск таких операндов и выдаётся их список как "принципиально

* Для бóльшей гибкости возможно использование переменных, допускающих многократное изменение значений (их имена должны начинаться с заданного префикса - последовательности символов, определяемых переменной *attrVar* секции [RuleVarData] файла настроек *data_flow.ini* (от 1 до 5 любых символов, по умолчанию это единичный символ "\$"); так как такие переменные *не следуют правилам однократного присваивания*, использовать их следует с осторожностью (удобно, напр., применять их в качестве аккумулятора при суммировании величин и др., при этом можно существенно "сэкономить" на числе переменных).

неопределённых" - во избежание "зависания" программы из-за напрасного ожидания определения этих переменных.

Входной коммутатор K_IN выбирает из ПИ инструкцию/и, все операнды которой/ых помечены флагом "Готов" (т.е. были вычислены ранее или им присвоено значение; в начале выполнения последовательности инструкций все флаги готовности обнулены!). По адресам операндов K_IN производит выборку их значений из ПД и передаёт для выполнения свободным АИУ. Время выполнения каждой инструкции отдельно задаётся в файле DATA_FLOW.INI (секция [Ticks]) в единицах тактов (*тиков*, допустимо задавать время выполнения нулевым), а время каждого такта - в *миллисек* (для корректного моделирования время такта нецелесообразно задавать не менее $\approx 10^{-2}$ сек, а общее время выполнения инструкций - порядка единиц секунд); длительность тика задаётся в переменной [Interval] секции [Tick_Interval]. Время вы́борки инструкций из ПИ и доставки к АИУ принимается (в этой версии) нулевым (ничто не мешает, однако, интерпретировать это время как часть продолжительности выполнения инструкций).

5. АИУ выполняет переданную ему инструкцию и результат передаёт **выходному коммутатору K_OUT**, который:

- а). Записывает результат операции по заданному адресу (имени) в ПД.
- б). Просматривает адреса операндов инструкций в ПИ и при совпадении выставляет флаг готовности у соответствующего операнда.
- д). "Готовые к выполнению" (по условию готовности всех входных операндов) инструкции (далее ГКВ) добавляются в *буфер инструкций*, из которого и выбираются для исполнения на свободных АИУ в соответствии с приоритетом (максимальный приоритет - первыми).

Процесс вычислений начинается, когда хотя бы одной (нескольким) ячейкам ПД присвоено (с помощью инструкций SET, см. список инструкций ниже) значение – тут же будут установлены флаги готовности операндов у одной или нескольких инструкций в ПИ, они будут выбраны в буфер и в соответствии с приоритетом исполнены на свободных АИУ (если последних не хватает – подождут в буфере), с результатами вычислений произойдёт то же самое... *процесс пошёл!* Постепенно у всё большего количества инструкций будут установлены ГКВ-флаги и число работающих АИУ будет возрастать... а по мере приближения к концу вычисления – начнёт падать (обычно именно так и бывает). Концом выполнения программы в данном случае считается однократное выполнение ВСЕХ инструкций. Проблемы (при физической реализации) могут возникнуть, если в некоторый момент число ГКВ-инструкций превысит размер буфера (часть инструкций может пропасть); для предотвращения такой ситуации система слежения за наполненностью буфера принимает меры, позволяющие избежать потерь.

Существующие ограничения симулятора:

1. Поддерживаются только линейные алгоритмы (условное выполнение реализуется *предикатным способом*).
2. Требуется следование концепции "однократного присваивания" (любое вновь вычисляемое значение необходимо поместить в *новую ячейку памяти*; однако см. ранее описанное *исключение*). Согласно концепции *однократного присваивания* содержимое ПД изменять нельзя; ежели в качестве адреса результата выполнения инструкции указан уже имеющийся в ПД адрес, будет выдано предупреждение (выбор пользователем варианта продолжения приведёт к перезаписи содержимого ПД по данному адресу со значимой вероятностью неправильного решения... *однако пользователь предупреждён!*); в противном случае объект данных в ПД будет создан с присвоением ему соответствующего значения.
3. Сейчас поддерживаются арифметические трехадресные (*два входных операнда плюс результат*) и двухадресные (*один входной операнд и результат*) инструкции, все над вещественными 64-х битовыми переменными типа *double*; точность представления данных - около 16 десятичных цифр). Все адреса – *прямые* (последовательность символов в поле адреса прямо указывает на данные в ПД); *регистры в явном виде отсутствуют*.

Для занесения исходных для расчёта данных в ПД (*инициализация данных*) используется инструкция SET (выполняется коммутатором внепроцессорно); для копирования (“размножения”) данных удобно использовать CPY. *Нет проблем сколь угодно расширить число выполняемых инструкций... чай, с симулятором-с работаем!*

Список инструкций приведён в таблице ниже (каждая записывается в отдельной строке, после *мнемоники* (трёхсимвольное краткое обозначение инструкции; допускается произвольный регистр символов, но при считывании принудительно устанавливаются заглавные буквы) инструкции обязателен пробел, в качестве разделителей адресов операндов и результата может использоваться запятая или пробел, в качестве адресов можно использовать последовательность любых символов в количестве до выше определённого числа (*имена операндов регистрозависимы*), после символа “;” располагается комментарий, строки файла программы с начальными “;” и пустые игнорируются и не считываются в ПИ).

Все инструкции выполняются *предикатно*, т. е. исполняются только в случае значения последнего в списке параметра true (отсутствие этого поля в исходной программе также соответствует true), в случае false инструкция не выполняется. Это статический вариант, позволяющий иметь полную совместимость по исходному коду с вариантом без предикатного варианта синтаксиса (поле предиката у выполняемых инструкций оставляется пустым).

При динамическом варианте в поле флага-предиката находится бóлева переменная (в списке инструкций описана как [а_флагП], где обозначение в

квадратных скобках соответствует необязательности наличия ея), формально являющаяся обычной переменной, подчиняющейся всем правилам обращения с переменными в DATA-FLOW вычислителе, но принимающей только два значения – true или false, определяемыми инструкциями предикатов (см. конец нижеследующей таблицы). Перед именем этой переменной могут находиться символы ‘!’ или ‘~’, означающий инверсию логического значения переменной. Для инструкции, содержащей поле предиката, признаком “готовности к выполнению” является, кроме “готовности” всех операндов, ещё и состояние true предиката (статические true/false или соответствующее состояние переменной в ПД). Для статического варианта возможен даже изврат вида !false/~false) и !true/~true); в этом случае при считывании файла инструкций препроцессор конвертирует эти значения в true и false соответственно.

Инструкция SET и все инструкции-предикторы (находящиеся в нижней части таблицы и начинающиеся с символа ‘Р’, далее будем использовать термины “Р-операции” или “не Р-операции”). Эти действия исполняются по условию “готовности” всех (1 или 2-х) операндов данной инструкции. “Не Р-инструкция” исполняются при условии готовности всех операндов (1 или 2-х) и значении true флага-предиктора (иногда полезно рассматривать его как дополнительный булев операнд). Собственно “Р-операторы” воспринимают операнды в виде логических значений (“ноль” или “не-ноль” как false/true соответственно) и результат возвращают в виде логического значения. И, конечно, результат исполнения каждого оператора вызывает установку соответствующего (по имени) операнда всех иных (кроме данной) инструкций (включая булево значение флага-предиката), что инициирует выполнение целого ряда операций (по условию ГКВ). У нескольких (хотя бы двух) операторов с противоположными значениями флага-предиката могут быть одинаковые имена результата (т.к. физически выполнится только один оператор с конечным значением флага-предиката true).

Из таблицы видно, что частично инструкции дублируются (часть имеет наклонность “арифметическую”, а часть – “логическую”).

№№	Инструкция	Выполняемое действие
1	SET вещ_число, а_результат	Вещественное число (<u>разделитель целой и дробной части - десятичная точка</u>) помещается в ПД по адресу а_результат. Внимание! Это единственная инструкция, <u>принимáющая число как операнд</u> (если символы операнда не могут быть интерпретированы числом, выдаётся предупреждение).
2	CPY а_операнд, а_результат [,а_флагП]	Значение по адресу а_операнд копируется по адресу а_результат; действие производится только при а_флагП=TRUE
3	ABS а_операнд, а_результат [,а_флагП]	Модуль значения по адресу а_операнд копируется по адресу а_результат
4	SGN а_операнд, а_результат [,а_флагП]	По адресу а_результат заносится (+1,0) при положительном (и нулевом) значении числа по адресу а_операнд, в противном случае заносится (-1,0)

5	NEG а_операнд, а_результат [,а_флагП]	По адресу а_результат заносится значениям числа по адресу а_операнд с изменённым на противоположный знаком
6	CEL а_операнд, а_результат [,а_флагП]	По адресу а_результат заносится округлённое вверх значениям числа по адресу а_операнд (наименьшее целое, не меньшее а_операнд)
7	FLR а_операнд, а_результат [,а_флагП]	По адресу а_результат заносится округлённое вниз значениям числа по адресу а_операнд (наибольшее целое, не большее а_операнд)
8	RND а_операнд-1, а_операнд-2, а_результат [,а_флагП]	В а_результат помещается (<i>псевдо</i>)случайное число из диапазона [а_операнд-1 ÷ а_операнд-2]
9	ADD а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Значения вещественных чисел по адресам операндов а_операнд-1 и а_операнд-2 складываются и результат помещается по адресу а_результат
10	SUB а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Из вещественного числа по адресу а_операнд-1 вычитается число по адресу а_операнд-2 и результат помещается по адресу а_результат
11	MUL а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Значения вещественных чисел по адресам операндов а_операнд-1 и а_операнд-2 перемножаются и результат помещается по адресу а_результат
12	DIV а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Вещественное число по адресу а_операнд-1 делится на число по адресу а_операнд-2 и результат помещается по адресу а_результат
13	DQU а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Число по адресу а_операнд-1 делится на число по адресу а_операнд-2 и частное от деления нацело помещается по адресу а_результат (перед действием над обоими операндами производится операция floor, напр. floor(2.3)=2 , floor(3.8)=3 , floor(-2.3)=-3 , floor(-3.8)=-4)
14	DRE а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Число по адресу а_операнд-1 делится на число по адресу а_операнд-2 и остаток от деления нацело помещается по адресу а_результат (операнды преобразуются так, как описано в операции DQU)
15	SQR а_операнд, а_результат [,а_флагП]	Вычисляется квадратный корень из вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
16	POW а_операнд-1, а_операнд-2, а_результат [,а_флагП]	Вещественное число по адресу а_операнд-1 возводится в степень вещественного по адресу а_операнд-2 и результат помещается по адресу а_результат
17	SIN а_операнд, а_результат [,а_флагП]	Вычисляется синус из вещественного числа по адресу а_операнд (в радианах) и результат помещается по адресу а_результат
18	COS а_операнд, а_результат [,а_флагП]	Вычисляется косинус из вещественного числа по адресу а_операнд (в радианах) и результат помещается по адресу а_результат
19	TAN а_операнд, а_результат [,а_флагП]	Вычисляется тангенс из вещественного числа по адресу а_операнд (в радианах) и результат помещается по адресу а_результат

20	ASN а_операнд, а_результат [,а_флагП]	Вычисляется арксинус из вещественного числа по адресу а_операнд и результат помещается по адресу а_результат (в радианах)
21	ACN а_операнд, а_результат [,а_флагП]	Вычисляется арккосинус из вещественного числа по адресу а_операнд и результат помещается по адресу а_результат (в радианах)
22	ATN а_операнд, а_результат [,а_флагП]	Вычисляется арктангенс из вещественного числа по адресу а_операнд и результат помещается по адресу а_результат (в радианах)
23	LOG а_операнд, а_результат [,а_флагП]	Вычисляется натуральный логарифм из вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
24	EXP а_операнд, а_результат [,а_флагП]	Вычисляется экспонента e^X от вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
25	SNH а_операнд, а_результат [,а_флагП]	Вычисляется гиперболический синус $(e^X - e^{-X})/2$ от вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
26	CNH а_операнд, а_результат [,а_флагП]	Вычисляется гиперболический косинус $(e^X + e^{-X})/2$ от вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
27	TNH а_операнд, а_результат [,а_флагП]	Вычисляется гиперболический тангенс $(e^X - e^{-X}) / (e^X + e^{-X})$ от вещественного числа по адресу а_операнд и результат помещается по адресу а_результат
<i>Инструкции, устанавливающие флаги предикатов согласно заданным условиям (операнд – вещественное число, результат – логическое значение)</i>		
28	PGE а_операнд1, а_операнд2, а_флагП	При условии а_операнд-1 >= а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе – FALSE
29	PLE а_операнд-1, а_операнд-2, а_флагП	При условии а_операнд <= а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе - FALSE
30	PGT а_операнд-1, а_операнд-2, а_флагП	При условии а_операнд > а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе – FALSE
31	PLT а_операнд-1, а_операнд-2, а_флагП	При условии а_операнд < а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе – FALSE
32	PEQ а_операнд-1, а_операнд-2, а_флагП	При условии а_операнд == а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе - FALSE
33	PNE а_операнд-1, а_операнд-2, а_флагП	При условии а_операнд != а_операнд-2 флаг а_флагП устанавливается в TRUE, иначе - FALSE
<i>Инструкции, оперирующие с предикатами (один операнд и результат – логические значения)</i>		
34	PNT а_флагП, а_результат_флагП	Отрицание: возвращается TRUE при а_флагП=FALSE и FALSE при а_флагП=TRUE; общепринятые символы операции: “NE”, “NOT”, “¬”
<i>Инструкции, оперирующие с предикатами (оба операнда и результат – логические значения)</i>		

35	PAN a_флагП1, a_флагП2, a_результат_флагП	Конъюнкция предикатов: возвращается TRUE при значении TRUE обоих операндов, в противном случае возвращается FALSE; общепринятые символы операции: “И”, “AND”, “ \wedge ”
36	POR a_флагП1, a_флагП2, a_результат_флагП	Дизъюнкция предикатов: возвращается TRUE при значении TRUE хотя бы одного из двух операндов, в противном случае возвращается FALSE; общепринятые символы операции: “ИЛИ”, “OR”, “ \vee ”
37	PIM a_флагП1, a_флагП2, a_результат_флагП	Импликация предикатов (операция “ <i>слéдование</i> ”): возвращается FALSE при a_флагП1=TRUE “И” a_флагП2=FALSE, в противном случае возвращается TRUE; общепринятый символ операции: “ \rightarrow ”
38	PRM a_флагП1, a_флагП2, a_результат_флагП	Обратная импликация предикатов (операция “ <i>обратное слéдование</i> ”): возвращается FALSE при a_флагП1=FALSE “И” a_флагП2=TRUE, в противном случае возвращается TRUE; общепринятый символ операции: “ \leftarrow ”
39	PEV a_флагП1, a_флагП2, a_результат_флагП	Эквиваленция (эквивалентность): возвращается TRUE при ((a_флагП1=TRUE “И” a_флагП2=TRUE) “ИЛИ” (a_флагП1=FALSE “И” a_флагП2=FALSE)), в противном случае возвращается FALSE; общепринятые символы операции: “XNOR”, “ \leftrightarrow ”, “ \equiv ”
40	PXR a_флагП1, a_флагП2, a_результат_флагП	Исключающее ИЛИ (операция “ <i>строгая дизъюнкция</i> ”): возвращается FALSE при ((a_флагП1=TRUE “И” a_флагП2=TRUE) “ИЛИ” (a_флагП1=FALSE “И” a_флагП2=FALSE)), в противном случае возвращается TRUE; общепринятые символы операции: “XOR”, “ \oplus ”
41	PAP a_флагП1, a_флагП2, a_результат_флагП	Стрелка Пирса (логическое “ <i>ИЛИ-НЕ</i> ”): возвращается TRUE при (a_флагП1=FALSE “И” a_флагП2=FALSE), в противном случае возвращается FALSE; общепринятые символы операции: “ИЛИ-НЕ”, “NOR”, “ \downarrow ”
42	PNS a_флагП1, a_флагП2, a_результат_флагП	Штрих Шеффера (логическое “ <i>И-НЕ</i> ”): возвращается FALSE при (a_флагП1=TRUE “И” a_флагП2=TRUE), в противном случае возвращается TRUE; общепринятые символы операции: “И-НЕ”, “NAND”, “ \uparrow ”

В целом используются 3 типа инструкций (операторов) – SET предназначена для задания исходных данных, арифметические операторы (А-операторы) переназначены для выполнения арифметических, тригонометрических и иных действий в основном с числами с плавающей запятой и предикатные операторы (Р-операторы, назначение которых назначение которых – производить логические операции (на вход могут поступать и “плавающие” числа, которые интерпретируются как false при нулевом и true при любом ненулевом значении, результат Р-операции всегда 0 или 1, интерпретируемые как false и true соответственно).

Как видно из нижерасположенной таблицы, поле переменных едино и результат выполнения как А-операторов, так и Р-операторов имеют возможность изменять любой из операндов А- и Р-операторов. Программист может на логическом уровне разграничить области “плавающих” и “булевых” переменных путём добавления,

например, в начало последних одного из выразительных для него символов - например, **b** (от *bool*) в качестве квалификатора.

Вследствие сказанного отличие А- от Р-операторов только одно – при каждом вычислении Р-оператора результат вычисления сравнивается с полем флага-предиката *всех* А-операторов (Р-операторы не учитываются, т.е. не могут выполняться условно) и разрешает или запрещает (даже при готовности всех операндов А-оператора) его выполнение (при значении флага-предиката *true* или с учётом лидирующих символов в имени поля флага предиката ‘!’ или ‘~’ для отрицания соответственно). Напомним, что в данной модели каждый оператор может выполняться только *единокровно*.

Т.о. флаг-предикат выполняет функции одного из входных операндов и при анализе динамики выполнения (установка в ненулевое значение переменной *needDrawColors* в секции [DrawColorExec] файла настроек системы DATA_FLOW.INI) в случае значения *true* соответствующая ячейка таблицы окрашивается в заданный цвет (по умолчанию *зелёный*, что ассоциируется с соответствующим *салатовым* в ячейках классических операндов).

Группы операторов	Допустимый приёмник результата выполнения “родительского” оператора	Идентификатор канала передачи данных
1. Определение исходных данных для расчёта (единственный пример - SET)	1. Единственный операнд (<i>float</i>) А-оператора	10
	2. Любой из двух операндов (<i>float</i>) А-оператора	20
	3. Единственный операнд (<i>float</i>) П-оператора	11
	4. Любой из двух операндов (<i>float</i>) П-оператора	21
2. А-операторы (арифметические операторы, 1 или 2 операнда + возможно, флаг-предикат)	1. Единственный операнд (<i>float</i>) А-оператора	100
	2. Любой из двух операндов (<i>float</i>) А-оператора	200
	3. Единственный операнд (<i>float</i>) П-оператора	101
	4. Любой из двух операндов (<i>float</i>) П-оператора	201
3. Р-операторы (предикатные операторы, 1 или 2 операнда)	1. Единственный операнд (<i>float</i>) А-оператора	110
	2. Любой из двух операндов (<i>float</i>) А-оператора	210
	3. Единственный операнд (<i>bool</i>) П-оператора	111
	4. Любой из двух операндов (<i>bool</i>) П-оператора	211

Формат оператора с использованием предиката приведён ниже (квадратные скобки означают необязательность заключённого в них текста) :

ADD Op1 [, Op2], Res [: Pred] [: комментарий до конца строки]

где ADD – мнемоника команды (здесь - *сложение*), Op1 и Op2 – аргументы команды, Res – результат выполнения команды, Pred – необязательное имя предиката,

символом-разделителем перед полем предиката могут служить ':' или ',', перед именем предиката допускается символ отрицания '!'.
При выявлении проблем с вычислениями полезен анализ флагов, управляющих вычислениями (5-я слева колонка окна "Память инструкций"). В этой колонке индицируется состояние 5-х булевых флагов в форме |a1|a2|t|b|c|d| (в случае неиспользования флага в конкретной ситуации выдается 'x').

Установка флагов 'a1' и 'a2' индицирует *признак готовности операндов* (т.е. значение по соответствующему адресу в "Памяти данных" присвоено инструкцией SET или вычислено предыдущими инструкциями), флаг 't' соответствует значению флага предиктора true (т.е. данную инструкцию выполнять разрешено – при готовности иных операндов, конечно). Флаг 'b' устанавливается при добавлении данной инструкции в буфер, флаг 'c' устанавливается на время исполнения инструкции (и очищается при окончании оно), флаг 'd' устанавливается после выполнения инструкции.

При “зависании” процесса вычислений (прекращена выборка операций) несложно по состоянию этих флагов определить причину - обычно это инструкция, один (или оба) флага 'a1' и 'a2' готовности операндов которой сброшены; в этом случае рациональным будет анализ флага 'd' у инструкции, адрес результата которой равен адресу (адресам) соответствующих операндов проблемной инструкции (и, конечно, наличие соответствующей строки в ПД).

Для анализа и отладки программы предназначен вариант “Анализ/отладка” главного меню, причём при выборе “Невыполнившиеся инструкции” визуализируются (цветом) невыполнившиеся (на данный момент) инструкции (по ненулевому значению флага 'd'), “Неиспользованные результаты” – команды, результат выполнения которых не используется ни одной иной инструкцией в качестве операнда, “Неопределённые операнды” – операнды команд, не совпадающие с результатом выполнения ни одной другой инструкции.

Дополнительно удобно воспользоваться функцией связи “Результат ↔ Операнды”, активизируемой посредством всплывающего меню при щелчке правой кнопкой мыши при нахождении курсора над таблицей “Память команд”. Установив фокус на ячейке с выбранным адресом в столбце “Результат” и выбрав из выпадающего меню вариант “Результат → Операнды”, получим индикацию цветом всех ячеек операндов, имеющих адрес, совпадающий с адресом выбранного результата. Вариант “Операнд → Результат” этого же меню позволяет выполнить обратное действие – выяснить, от результата выполнения какой инструкции зависит значение выбранного входного операнда. Информация о связях выдаётся также в окно протоколирования и файл *.PRO (признак – в начале и конце строки символы -t-). Информация о каждой из зависимых от результата данной по операндам инструкций выдаётся последовательно в форме N/M(A|B), где N – номер инструкции в таблице “Память команд”, M – идентификатор канала передачи данных (см. табл. выше), A – номер входного операнда, B – общее количество входных операндов в инструкции.

Заметим, что при работе симулятора при выполнении каждой инструкции в окно протоколирования (и файл *.PRO) выдаются аналогичные данные.

В базовом варианте симулятора число АИУ, инструкций в ПИ, пар “адрес/значение” в ПД и размер буфера не могут превышать 10^6 элементов.

Далее рассматривается *несложная задача* нахождения корней x_1, x_2 квадратного уравнения (КВУ) $a \times x^2 + b \times x + c = 0$ (a, b, c - заданные константы).

Широкоизвестно (*), что корни одного КВУ суть $x_{1,2} = (-b \pm \text{SQR}(b^2 - 4 \times a \times c)) / (2 \times a)$. Числа a, b, c поместим в ячейки A, B, C соответственно, константы 2, 4 и (-1) - в ячейки TWO, FOUR и NEG_ONE. Теперь пишем с помощью любого текстового редактора программу (файл SQUA_EQU_2.SET):

```
; Решение полного квадратного уравнения в вещественных числах
; A x X^2 + B x X + C = 0
; in case A = 1, B = 7, C = 3 the solve is: X1 = -0.45862 , X2 = -6.5414
;
MUL A, TWO, A2 ; 2 x A -> A2
MUL A, FOUR, A4 ; 4 x A -> A4
MUL B, NEG_ONE, B_NEG ; NEG_ONE x B -> B_NEG
POW B, TWO, BB ; B^2 -> BB
MUL A4, C, AC4 ; A4 x C -> AC4
SUB BB, AC4, D ; BB - AC4 -> D [discriminant]
SQR D, sqrt_D ; sqrt(D) -> sqrt_D ; квадратный корень из D
;
ADD B_NEG, sqrt_D, W1 ; B_NEG + sqrt_D -> W1
SUB B_NEG, sqrt_D, W2 ; B_NEG - sqrt_D -> W2
DIV W1, A2, X1 ; W1/A2 -> X1
DIV W2, A2, X2 ; W2/A2 -> X2
;
SET 1.0, A ; 1.0 -> A
SET 7.0, B ; 7.0 -> B
SET 3.0, C ; 3.0 -> C
;
SET 2, TWO ; 2 -> TWO
SET 4, FOUR ; 4 -> FOUR
SET -1, NEG_ONE ; (-1) -> NEG_ONE
```

Условное выполнение операторов (метод предикатов) демонстрируется программой SQUA_EQU_2.PRED.SET (здесь единственная переменная-предикат IS_re выделена красным):

```
; Решение полного квадратного уравнения
; для получения решения в мнимых числах используем флаг предиката
; IS_re есть true при значении дискриминанта D >= 0 (вещественные корни)
; A x X^2 + B x X + C = 0
```

* Найдённые древние вавилонские глиняные таблички, датированные 1800 ÷ 1600 годами до н.э., являются самыми ранними свидетельствами изучения квадратных уравнений (КВУ). На этих же табличках изложены методы решения некоторых типов КВУ. Древнеиндийский математик *Брахмагута* в VIII столетии до н.э. впервые использовал КВУ в форме $ax^2=c$ и $ax^2+bx=c$, а также привёл методы решения этих КВУ. Вавилонские математики примерно с IV века до н.э. и китайские математики со II века до н.э. использовали метод дополнения квадрата для решения КВУ с положительными корнями. Однако только около 300 года до н.э. *Эвклид* придумал более общий геометрический метод решения КВУ.

Первым математиком, который нашёл решения КВУ с отрицательными корнями в виде алгебраической формулы, был *Брахмагута* (Индия, VII столетие н.э.).

```

; in case A = 1, B = 7, C = 3 the solve is: re_X1=-0.45862,im_X1=0; re_X2=-6.5414,im_X2=0
; in case A = 1, B = 3, C = 3 the solve is: re_X1=-1.5,im_X1=0.866; re_X2=-1.5,im_X2=-0.866
;
MUL A, TWO, A2, !false ; 2×A→A2
MUL A, FOUR, A4 ; 4×A→A4
MUL B, NEG_ONE, B_NEG ; NEG_ONE×B→B_NEG
POW B, TWO, BB ; B^2→BB
MUL A4, C, AC4 ; A4×C→AC4
SUB BB, AC4, D ; BB-AC4→D[iskriminant]
;
SQR D, sqrt_D: !S_re ; sqrt(D)→sqrt_D
ADD B_NEG, sqrt_D, W1: !S_re ; B_NEG+D_SQRT→W1
SUB B_NEG, sqrt_D, W2: !S_re ; B_NEG-D_SQRT→W2
DIV W1, A2, re_X1: !S_re ; W1/A2→re_X1
DIV W2, A2, re_X2: !S_re ; W2/A2→re_X2
;
MUL D, NEG_ONE, NEG_D: !S_re ; NEG_ONE×D→NEG_D
SQR NEG_D, sqrt_D: !S_re ; sqrt(NEG_D)→sqrt_D
DIV B_NEG, A2, re_X1: !S_re ; 1-th root (real)
DIV sqrt_D, A2, im_X1: !S_re ; 1-th root (img)
CPY re_X1, re_X2: !S_re ; 2-th root (real)
DIV sqrt_D, A2, W: !S_re ; temp for im_X2
MUL W, NEG_ONE, im_X2: !S_re ; 2-th root (im)
;
SET 1, A ; 1→A
SET 3, B ; 7/3→B (re/im)
SET 3, C ; 3→C
;
SET 2, TWO ; 2→TWO
SET 4, FOUR ; 4→FOUR
SET -1, NEG_ONE ; (-1)→NEG_ONE
;
SET 0,ZERO
;
PGE D,ZERO, !S_re ; !S_re←true if D>=0

```

При старте симулятора из файла DATA_FLOW.EXE в него автоматически загружается программа из файла настроек DATA_FLOW.INI. Нажатием Ctrl+F4 в среде симулятора вызывается текстовый редактор с именем, занесённым в строку File настроечного файла DATA_FLOW.INI; Shift+F4 позволяет таким же образом редактировать DATA_FLOW.INI. Для обновления ПИ и файла настроек служит клавиша F5 (это при MODE=0 в DATA_FLOW.INI, при MODE=1 редактор вызывается *модально* и обновление происходит автоматически по событию закрытия окна редактора).

Протоколы расчёта сохраняются при нажатии пользователем клавиши F2 в файлах в подкаталоге (относительно текущего каталога) Out!Data с расширением “txt” и началом имени файла, приведёнными ниже; после символа ‘!’ содержатся данные об имени проекта, текущих настройках и данными времени выполнения программы моделирования):

1. Начало имени файла PRO! - главный файл протокола (представляет собой копию строк, выводимых при решении задачи в *верхний фрейм* главного окна программы-симулятора).
2. Начало имени файла TPR! - файл анализа времени загрузки АИУ инструкциями программы. *Именно этот файл представляет собой план (расписание) выполнения параллельной программы по номерам АИУ (если параметр – первый столбец) или по времени начала выполнения операторов (тогда параметр суть второй столбец).*

Формат каждой из основной части строк этого файла (числовые данные выровнены по *правой границе поля*):

- 10 символов: номер (начиная с 0) АИУ, выполняющего данную инструкцию (целое).
- 10 символов: время (начиная с момента начала вы́борки инструкций) начала выполнения данной инструкции данным АИУ), тактов (целое).
- 10 символов: время (начиная с момента начала вы́борки инструкций) конца выполнения данной инструкции данным АИУ), тактов (целое).
- 10 символов: время выполнения данной инструкции данным АИУ, тактов (целое).
- 10 символов: номер (начиная с 0) инструкции, выполняемой данным АИУ (целое).
- До конца строки: текст инструкции (в квадратных скобках); после адреса/идентификатора в фигурных скобках выдаётся значение по данному адресу в ПД (*операнд* или *результат*) + (неизменный) комментарий.

3. Начало имени файла TST! – файл количества одновременно исполняемых инструкций (*) в функции времени; эти данные удобно использовать путём импорта их в MS Excel (**) для построения зависимости числа исполняемых инструкций от времени (числовые данные сформатированы по *правой границе поля*):

- 10 символов: время с начала выполнения программы, тактов (целое).
- 10 символов: число АИУ, занятых выполнением инструкций (целое).
- Начиная с 26-й позиции перечисляются (через 2 пробела) все выполняемые в данный момент инструкции в формате “*Номер_АИУ /Номер_инструкции_в_ПИ/Мнемоника инструкции*”.

4. Начало имени файла DAT! - файл вычисленных программой данных (копия строк *правого фрейма* главного окна). Формат каждой строки этого файла (данные сформатированы по *правой границе поля*):

- 20 символов: адрес/идентификатор переменной.
- 20 символов: значение по этому адресу (вещественное).
- 5 символов: пробелы.
- До конца строки: номер *N* инструкции в виде #*N* и через пробел полный (включая комментарии) текст инструкции (в квадратных скобках), в результате выполнения которой получено данное значение.

* При задании количества АИУ (величина `max_Procs` в файле настроек `DATA_FLOW.INI`), заведомо большим (или равным) максимальному числу выполняемых одновременно инструкций, может быть определён (априори неизвестный) потенциал распараллеливания данной программы.

** Для получения такого графика следует провести расчёт в симуляторе и сохранить файлы протокола расчёта (*F2* или “*Файлы → Сохранить протоколы расчёта*”); для импорта данных следует в MS Excel выполнить: “*Открыть → Выбрать нужный файл с расширением `tst` → Далее → Построить диаграмму по двум крайним слева столбцам* (первый столбец – ось абсцисс, второй столбец – ось ординат; тип диаграммы – *график* или *точечная*)”.

ВНИМАНИЕ! При каждом сохранении файлов протокола предыдущие версии файлов перезаписываются без сохранения предыдущих значений. Пользователь должен самостоятельно позаботиться о сохранении нужной ему информации!

В версиях данного симулятора начиная с 2018 г. включена возможность графического изображения функции ИНТЕНСИВНОСТИ ВЫЧИСЛЕНИЙ от параметра времени выполнения алгоритма (визуализируется в отдельном окне после *успешного выполнения программы* при значении параметра NeedAutoGraph#0 секции [Auto_Graph]) файла настроек DATA_FLOW.INI), в ином случае для визуализации графика необходимо нажатие клавиши F6. Этот график следует рассматривать вспомогательным при первоначальном анализе целевой функции, для тщательного анализа удобнее использовать метод построения с помощью Excel.

Для более тонкого анализа введена возможность анализа связности информационного графа алгоритма. При этом анализируются информационные зависимости типа “результат → операнды” для каждой инструкции (кроме тех, от результата выполнения которых не зависит ни один из операндов – т.е. это конечный результат выполнения программы). Данные выдаются в файл с расширением COI в нижеследующем формате (каждое значение – в 10 позициях строки).

1	20	0,741	74.074%
2	5	0.185	18.519%
3	2	0,074	7.407%

Показатель альтернативности = 0,350

Здесь первый столбец указывает зависимость – при значении 1 это отношение (связность) “1→1” (т.е. результат выполнения является операндом только для одной иной инструкции), значение 2 говорит о связности “1→2” и т.д. Второй столбец показывает, сколько всего таких отношений в данной программе, в столбцах 3 и 4 индицируется то же самое в долях единицы и %% соответственно; эти данные удобно использовать при построении гистограмм.

В случае отношения типа “1→1” альтернативность выполнения следующих за данной инструкций минимальна, при отношениях “1→2”, “1→3” и более альтернативность вырастает; поэтому в качестве показателя альтернативности применено отношение сумм элементов второго столбца кроме первой строки к соответствующей величине первой строки. Чем больше эта величина, тем (потенциально) эффективнее управление динамикой вычислений в DATA-FLOW вычислителе.

“Ясно видно всем”, что неравномерность числа “готовых к выполнению” (далее - ГКВ) операторов по времени велика... а можно ли (хоть как-то?) попытаться сгладить её (эту неравномерность)? Для этого надо научиться управлять процессом

(фактически управлять **ИБ - интенсивностью вычислений** – распределением по времени выполнения программы числа параллельно выполняющихся операций).

А **КАК УПРАВЛЯТЬ ИНТЕНСИВНОСТЬЮ ВЫЧИСЛЕНИЙ?** Тому, однако, следуют (нижеизложенные) пункты...

При выборке инструкций из буфера для исполнения на ИУ всегда первыми выбираются инструкции с *максимальным по буферу приоритетом* (а он зависит от определенных *параметров инструкции*... а каких – тому следуют пункты). В файле настроек DATA_FLOW.INI стратегию порядка выборки определяют две величины - how_Calc_Param и how_Calc_Prior:

☺ how_Calc_Param (рассчитывается как сумма параметров всех *тестируемых* инструкций – т.е. инструкций, *зависящих по входным операндам от результата выполнения данной инструкции*):

- при how_Calc_Param=0 все значения параметров инструкций одинаковы (фактически инструкции будут переданы ИУ в порядке “первым вошёл – первым вышел”),
- при how_Calc_Param=1 для значения параметра используется датчик случайных чисел,
- при how_Calc_Param=2 – время выполнения тестируемой (кроме SET...) инструкции,
- при how_Calc_Param=3 – число тестируемых (кроме SET...) инструкций, для которых результат выполнения данной является *хотя бы одним* из входных операндов,
- при how_Calc_Param=4 – число входных операндов, тестируемых (кроме SET...) инструкций, совпадающих по адресам с адресом результата выполнения данной,
- при how_Calc_Param=5 – если у тестируемой (кроме SET и данной) инструкции имеется N входных операндов, из которых M к данному моменту *уже готовы* и результат выполнения данной инструкции совпадает по адресу с K из (N-M) *неготовых* входных операндов, то параметр *полезности* тестируемой инструкции равен $K/(N-M)$; при $K=(N-M)$ получаем максимальный единичный вес тестируемой инструкции,
- остальные равносильны how_Calc_Param=0 (умолчание).

☺ how_Calc_Prior (служит для управления выборкой по минимальному или максимальному значению определяемого how_Calc_Param параметра: при how_Calc_Prior=0 значение приоритета равно значению параметра, при остальных – *обратной величине* этого параметра).

Для системы DATA-FLOW возможен режим запуска с командной строкой; причём в командной строке обязательны 4 параметра:

имя_Set-файла_число_ АИУ_ how_Calc_Param_ how_Calc_Prior

где _ - один или несколько пробелов.

Например, строкой `data_flow.exe_slau_2.set_15_0_1` запускается программа-симулятор потокового вычислителя с файлом-программой `slau_2.set` с числом АИУ 15 и значениями параметров `how_Calc_Param` и `how_Calc_Prior` соответственно 0 и 1.

Начиная с версии 4.2 ("зима 2017") в симулятор введён препроцессор, при этом исходный файл с расширением `set` преобразуется препроцессором в файл с тем же именем и расширением `set_PrP` и именно последний файл считывается в ПИ.

Синтаксис препроцессорной команды (макроса) `for` для работы с одномерными псевдомассивами приведён ниже (между элементами допускается любое количество пробелов):

```
for[i]=i1,i2,i3 { ; комментарий до конца строки
...инструкция #1
SET f(i), A | M[f(i)] ;
ADD A | M[f(i)], A | M[f(i)], A | M[f(i)] ;
...:
...инструкция #N
}
```

Описание синтаксиса макроса:

- `f(i)` – функция от переменной-индекса псевдомассива (в качестве индекса используется единичный буквенный символ, данном случае это `i`) или выражения над числами - поддерживаются основных операций: сложение (+), вычитание (-), умножение (*), деление (/), остаток от деления (%) и возведение в степень (^), набор стандартных функций: *sin, cos, tg, ctg, arcsin, arccos, arctg, arcctg, sh, ch, th, cth, exp, lg, ln, sqrt*, круглые и квадратные скобки любой вложенности,
- `i1,i2,i3` - целые числа, определяющие начальное, конечное значения изменения переменной индекса цикла и шаг одного изменения,
- `A` – простая переменная,
- `M[]` – псевдомассив (имена псевдомассивов должны начинаться с буквы, длина до 32 символов),
- символ `'|'` означает возможность выбора из вариантов синтаксиса подстрок слева и справа, поле предиката в данном макросе не используется.

Начиная с версии 2018 г. включена возможность импорта данных из системы DATA-FLOW в пакет SPF@home. По нажатию F7 в текущем каталоге создаётся файл информационного графа алгоритма в формате DOT (файл *.gv, где “*” – имя обрабатываемого файла программы), приемлемый для считывания системой SPF@home (номера операторов начинаются со значения StartNumb в секции [StartNumbOps] конфигурационного файла DATA_FLOW.INI (по умолчанию 100), в расположенных после символа ‘;’ комментариях показывается, каким именно операторам соответствуют их номера).

Пример содержимого файла SQUA_EQU_2.GV при значении StartNumb по умолчанию:

```
#
// Valery Bakanov research computer complex (2008 and further); e881e@mail.ru, http://vbakanov.ru/left_1.htm
# Total edges in this directed graph: 21
/* This file was automatically created thru program DATA_FLOW.EXE
   from the original data file SQUA_EQU_2.SET */
#
digraph SQUA_EQU_2 {
100 -> 109 ; [[100]MUL A,TWO,A2,true; 2 * A -> A2] -> [[109]DIV W1,A2,X1,true; W1 / A2 -> X1 // #9 | #109]
100 -> 110 ; [[100]MUL A,TWO,A2,true; 2 * A -> A2] -> [[110]DIV W2,A2,X2,true; W2 / A2 -> X2 // #10 | #110]
101 -> 104 ; [[101]MUL A,FOUR,A4,true; 4 * A -> A4] -> [[104]MUL A4,C,AC4,true; A4 * C -> AC4]
102 -> 107 ; [[102]MUL B,NEG_ONE,B_NEG,true; NEG_ONE * B -> B_NEG] -> [[107]ADD B_NEG,sqrt_D,W1,true; B_NEG + sqrt_D -> W1]
102 -> 108 ; [[102]MUL B,NEG_ONE,B_NEG,true; NEG_ONE * B -> B_NEG] -> [[108]SUB B_NEG,sqrt_D,W2,true; B_NEG - sqrt_D -> W2]
103 -> 105 ; [[103]POW B,TWO,BB,true; B^2 -> BB] -> [[105]SUB BB,AC4,D,true; BB - AC4 -> D[iskriminant]]
104 -> 105 ; [[104]MUL A4,C,AC4,true; A4 * C -> AC4] -> [[105]SUB BB,AC4,D,true; BB - AC4 -> D[iskriminant]]
105 -> 106 ; [[105]SUB BB,AC4,D,true; BB - AC4 -> D[iskriminant]] -> [[106]SQR D,,sqrt_D,true; sqrt(D) -> sqrt_D]
106 -> 107 ; [[106]SQR D,,sqrt_D,true; sqrt(D) -> sqrt_D] -> [[107]ADD B_NEG,sqrt_D,W1,true; B_NEG + sqrt_D -> W1]
106 -> 108 ; [[106]SQR D,,sqrt_D,true; sqrt(D) -> sqrt_D] -> [[108]SUB B_NEG,sqrt_D,W2,true; B_NEG - sqrt_D -> W2]
107 -> 109 ; [[107]ADD B_NEG,sqrt_D,W1,true; B_NEG + sqrt_D -> W1] -> [[109]DIV W1,A2,X1,true; W1 / A2 -> X1 // #9 | #109]
108 -> 110 ; [[108]SUB B_NEG,sqrt_D,W2,true; B_NEG - sqrt_D -> W2] -> [[110]DIV W2,A2,X2,true; W2 / A2 -> X2 // #10 | #110]
111 -> 100 ; [[111]SET 1.0,,A,true; 1.0 -> A] -> [[100]MUL A,TWO,A2,true; 2 * A -> A2]
111 -> 101 ; [[111]SET 1.0,,A,true; 1.0 -> A] -> [[101]MUL A,FOUR,A4,true; 4 * A -> A4]
112 -> 102 ; [[112]SET 7.0,,B,true; 7.0 -> B] -> [[102]MUL B,NEG_ONE,B_NEG,true; NEG_ONE * B -> B_NEG]
112 -> 103 ; [[112]SET 7.0,,B,true; 7.0 -> B] -> [[103]POW B,TWO,BB,true; B^2 -> BB]
113 -> 104 ; [[113]SET 3.0,,C,true; 3.0 -> C] -> [[104]MUL A4,C,AC4,true; A4 * C -> AC4]
114 -> 100 ; [[114]SET 2,,TWO,true; 2 -> TWO] -> [[100]MUL A,TWO,A2,true; 2 * A -> A2]
114 -> 103 ; [[114]SET 2,,TWO,true; 2 -> TWO] -> [[103]POW B,TWO,BB,true; B^2 -> BB]
115 -> 101 ; [[115]SET 4,,FOUR,true; 4 -> FOUR] -> [[101]MUL A,FOUR,A4,true; 4 * A -> A4]
116 -> 102 ; [[116]SET -1,,NEG_ONE,true; (-1) -> NEG_ONE] -> [[102]MUL B,NEG_ONE,B_NEG,true; NEG_ONE * B -> B_NEG]
}
```

При нажатии Ctrl+F7 дополнительно создаётся файл метрик вершин графа (операторов) *.mvr для считывания системой SPF@home, в котором находятся времена выполнения каждого оператора (копия данных из раздела Times конфигурационного файла DATA-FLOW.INI с информацией о соответствии операторов с их номерами); значение времени выполнения соответствует параметру “Times” (секция [TIMES] конфигурационного файла DATA_FLOW.INI) и может быть получено в системе SPF@home с помощью Lua-вызова GetMetricOpByName(*N*, “Times”), где *N* – номер вершины графа (оператора).

При выгрузке с сервера нового варианта программы инсталляционный файл системы будет сохранён во вновь образованном подкаталоге In!Data текущего каталога системы.

Начиная с апреля 2020 г. все ранее описанные файлы протоколов после окончания каждого расчёта автоматически сохраняются в подкаталоге Out!Data главного каталога системы. Уникальность имён файлов достигается включением в состав имени метки времени с точностью до мсек, расширение имени – txt; имя файла состоит из 4-х полей, разделитель – восклицательный знак. Пример имени файла:

pro!slau_2.set!AIU=123_Param=0_Prior=0!20-04-2020_11-55-08-163.txt

Первое поле может быть pro, tpr, dat, tst, coi, gv, mvr и отражает тип протокола (см. выше). Второе поле – имя файла программы (set-файла).

Третье поле – настройки симулятора Data-Flow вычислителя, при которых выполнялось моделирование. AIU, Param, Prior – значения числа АИУ и параметров how_Calc_Param, how_Calc_Prior соответственно.

Четвёртое поле – дата и время начала моделирования в формате “day-month-year_hour-min-sec-msec”. При автоматической обработке результатов большого количества экспериментов полное имя файла легко может быть разобрано на составляющие с помощью строки формата “%s!%s!AIU=%d_Param=%d_Prior=%d!%d-%d-%d_%d-%d-%d-%d.txt”.

Начиная с марта 2021 имеется возможность динамического перемещения строк фокуса в таблице машинных команд (фокус перемещается каждые pass_Counts выполненных команд); это даёт возможность определить, в какой области программы в данный момент происходит обработка данных. Значение pass_Counts задаётся в секции [Vizu_Dynamic] файла DATA_FLOW.INI. При pass_Counts=1 можно получить излишнее мелькание на экране, поэтому рационально задавать значения 3 и выше (при <=0 эффект теряется).

Начиная с августа 2021 введён *режим спекулятивного выполнения инструкций* (“режим Itanium'a) - инструкции А-типа исполняются, но результат выполнения теряется после завершения (режим устанавливается при ненулевом значении переменной SpeculateExec в секции [RuleSpeculateExec] файла настроек DATA_FLOW.INI). Следствием применения этого режима является одинаковое общее время выполнения программы независимо от входных данных. Естественно, в случае имени флага-предиката в А-инструкции, не совпадающего ни с одним из имён результатов выполнения Р-инструкции, данная А-инструкция не будет выполнена никогда.

Начиная с апреля 2022 программный модуль DATA_FLOW.EXE при работе использует некоторые данные с сервера разработчика, будьте готовы при первом запуске программы после сообщения брандмауэра Защитника Windows разрешить доступ указанного исполняемого файла к серверу разработчика.

Уважаемые г-да/сúдари/товарищи... и *просто хорошие люди*! Ежели у кого-то *возникнут более разумные идеи* по управлению интенсивностью вычислений в вычислителях архитектуры DATA-FLOW – готов “воду лить и ноги мыть” в смысле общения... задействуйте, наконец, свои “серые мозговье клетки” (как говаривал незабвенный Э.Пуаро в произведениях А.Кристи)...

Удачи! Ваши предложения по усовершенствованию симулятора и созданные программы прошу направлять на e881e@mail.ru. Автор тепло поблагодарит и подарит конфетку (возможно, *шоколадную*) ...