Принципы векторной обработки данных

При векторной обработке (рис. 4) командное слово *i* считывается из памяти команд MI в регистр команд векторного процессора. Отличительной особенностью векторного процессора является наличие в его системе команд *векторных команд* и средств поддержки их реализации.

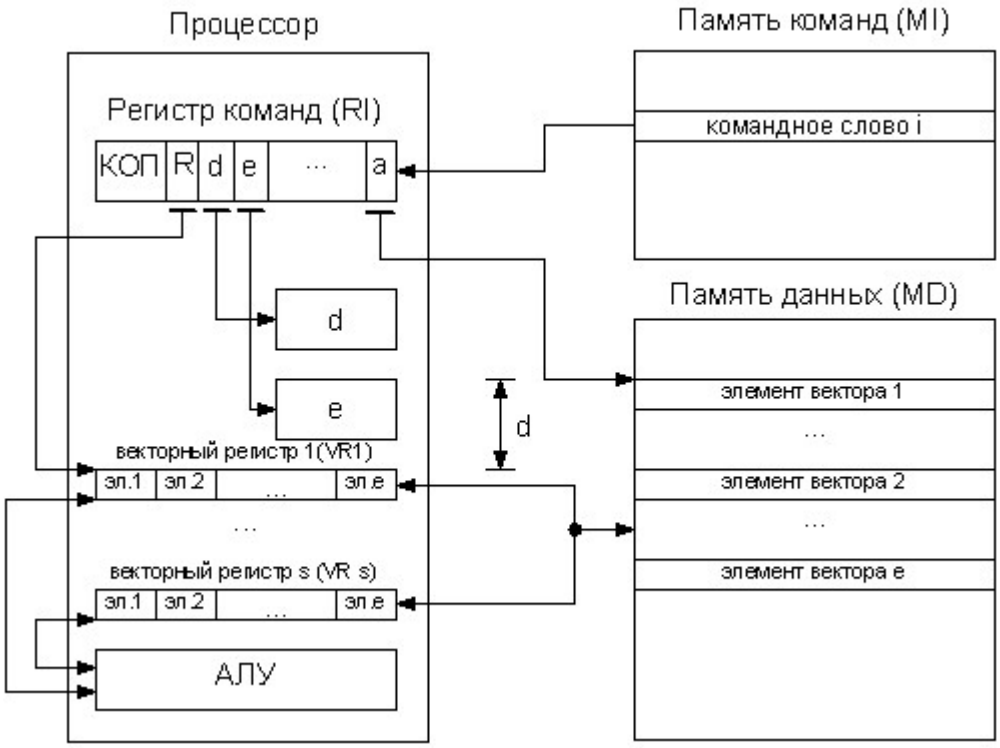


Рис. 4. Схема векторной обработки данных

Векторные команды предназначены для выполнения операций над векторными данными, которые представляют собой *e*-элементные наборы чисел, размещенные в памяти с определенной регулярностью и над которыми должны выполняться одни и те же действия (модель программирования ОКМД – одна команда, множество данных). Векторные данные имеют особо важное значение для реализации высокоскоростной числовой обработки, т. к. с помощью одной команды можно указать операцию над множеством данных, а в процессоре – приводить в действие различные механизмы параллельной обработки. В зависимости от типа и производителя процессоров указание командным словом векторных данных осуществляется по-разному, но при этом в любом случае должны быть заданы: адрес *a* базы векторных данных; значение приращения адреса *d* между элементами векторных данных; длина вектора *e*.

Для хранения считываемых из памяти векторных данных процессор должен иметь регистры, емкостью достаточной для вектора длины *e*. Такие регистры называются векторными. В зависимости от процессора количество векторных регистров в них варьируется, а также они сами могут быть различными по ёмкости и по **типу загрузки/выгрузки в/из них информации из/в памяти данных MD.** По последнему из названных критериев векторные регистры могут быть **однопортовыми**, в которых нельзя одновременно и записывать данные и читать их, или **двухпортовыми**, в которых можно одновременно и записывать данные и читать уже записанные данные из них (но разные элементы вектора).

При векторной обработке предварительные действия перед непосредственным выполнением операций требуют большого объема работ, связанных с начальной установкой различных регистров, например, длины вектора *e*, смещения *d*. Однако эти предварительные действия не предваряют выполнения всего цикла, как это происходит при скалярной обработке, и рассредоточены в стадиях выполнения векторных команд, но, тем не менее, для малых значений *e* эти действия могут привести к худшим показателям по сравнению со скалярной обработкой.

Преимуществом векторной обработки перед скалярной является то, что для выполнения одной и той же операции над e элементами вектора векторная команда выбирается из памяти только 1 раз. Именно этот фактор дает возможность одной командой инициировать *e*-кратное выполнение одной операции над различными данными. Таким образом, появляется принципиальная возможность эффективного использования конвейерных исполнительных устройств процессора.

Пусть память команд будет единой, а память данных разделена на 5 независимых по чтению и записи модулей.

Пример 1. **Векторный процессор** ***типа 1*** имеет один конвейер команд, 8 ***однопортовых*** векторных регистров, векторные исполнительные устройства – сумматор, умножитель, делитель конвейерного типа. Загрузка векторных регистров может осуществляться параллельно с выполнением арифметических операций, не связанных с загружаемым регистром.

Векторный процессор имеет следующие команды

VELD DATA, VR_X – поэлементная загрузка векторных данных (вектора) *DATA* в векторный регистр VR_X;

VEAD VR_X,VR_Y → VR_Z – поэлементное сложение содержимого векторного регистра VR_X с содержимым векторного регистра VR_Y и сохранение результата в векторном регистре VR_Z;

VESD VR_X, VR_Y → VR_Z – поэлементное вычитание из содержимого векторного регистра VR_X содержимого векторного регистра VR_Y и сохранение результата в векторном регистре VR_Z;

VEMD VR_X,VR_Y → VR_Z – поэлементное умножение содержимого векторного регистра VR_X на содержимое векторного регистра VR_Y и сохранение результата в векторном регистре VR_Z;

VEDD VR_X, VR_Y → VR_Z – поэлементное деление содержимого векторного регистра VR_X на содержимое векторного регистра VR_Y и сохранение результата в векторном регистре VR_Z;

VESTD VR_X – поэлементное сохранение содержимого векторного регистра VR_X в памяти данных.

Анализируя набор команд векторного процессора, необходимо отметить, что до выполнения любой операции над содержимым векторных регистров необходимые для этого данные (вектора) уже должны быть загружены в них. Следовательно, для выполнения арифметических операций необходимо **всегда выгружать операнды** из памяти данных, все действия в процессоре производятся над содержимым двух регистров.

В векторном процессоре возможно использование ***принципа зацепления***: действие, при котором результат исполнения команды не будет сохранен в каком-либо векторном регистре, а будет конвейерно поэлементно передаваться напрямую в исполнительное устройство, реализующее следующую по счёту команду, что позволяет ещё более ускорить вычисления.

Однако такая возможность накладывает определенные ограничения: команды, объединенные при **зацеплении**, являются **единой командой, и все конфликты должны проверяться на начало стадии *L* первой из них.**

Выполнение любой векторной команды разбивается на 7 стадий:

B – выборка команды из памяти команд MI;

H – проверка на бесконфликтность (виды конфликтов будут описаны ниже);

K – выдача команды;

D – дешифрация команды;

G – подготовка параметров: занесение значений *e* и *d* в соответствующие регистры;

L – настройка конвейера;

E – поэлементное выполнение операции.

Пусть стадии *B, H, K, D* требуют для своего исполнения 1 МТ каждая, G и L – по 2 МТ, загрузка из памяти или сохранение в память одного элемента вектора занимает 1 МТ, сложение/вычитание двух элементов векторов – 1 МТ, умножение – 3 МТ, деление – 6 МТ.

Это предполагает, что АЛУ векторного процессора такое же как было применено в суперскалярном, рассмотренном выше.

При анализе процесса выполнения программы векторным процессором и определения ее временных характеристик необходимо выполнить следующие действия:

1. Перевести текстовое выражение фрагмента программы в ассемблерный код в соответствии с системой команд нашего векторного процессора.
2. Построить временную диаграмму выполнения векторным процессором заданного фрагмента программы.

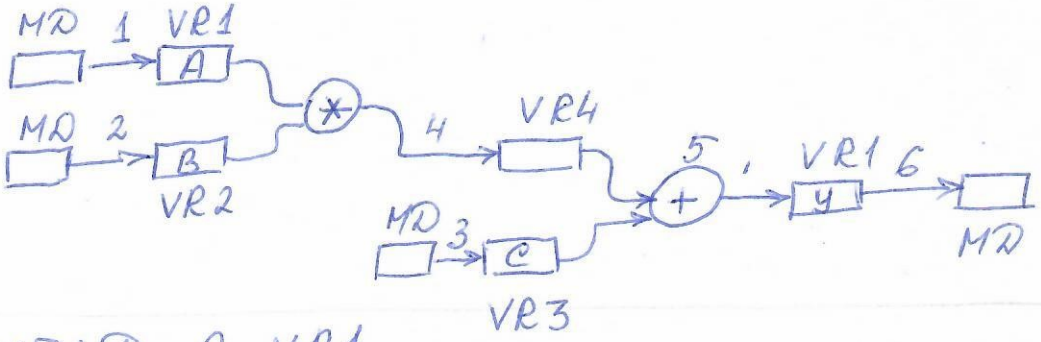
Далее рассмотрим пример выполнения фрагмента программы, рассмотренного при анализе суперскалярного процессора, векторным **процессором типа 1, с однопортовыми векторными регистрами.** Ассемблерный код для него представляет собой последовательность из 18 векторных команд (см. таблицу 2), каждая из которых будет исполнена только 1 раз.

$y = A \overset{①}{*} B \overset{②}{+} C$ A, B, C, y – векторы длины N

① Перевод в ассемблерный код

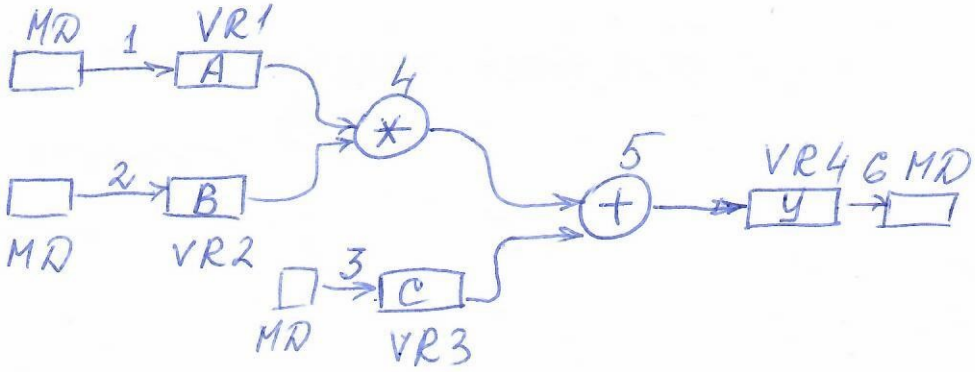
I

1. VELD A, VR1
2. VELD B, VR2
3. VELD C, VR3
4. VEMD VR1, VR2 → VR4
5. VEAD VR3, VR4 → VR1
6. VESTD VR1



II

1. VELD A, VR1
2. VELD B, VR2
3. VELD C, VR3
4. VEMD VR1, VR2 → Σ
5. VEAD *, VR3 → VR4
6. VESTD VR4



For I=1 to *N* ***DO***
S(i)= Z(i) / (X (i) – Q(i)) + P(i)*Y(i)/U(i) + (W(i)-V(i))*L(i).

Таблица 2

Программа для векторного процессора типа 1

Номер векторной команды	**Команда**
1	VELD X,VR1
2	VELD Q,VR2
3	VELD P,VR3
4	VELD Y,VR4
5	VESD VR1,VR2 → VR5=*(X-Q)*
6	VEMD VR3,VR4 → VR6=*(P*Y)*
7	VELD W,VR7
8	VELD V,VR8
9	VESD VR7,VR8 → VR1=*(W-V)*
10	VELD Z,VR2
11	VELD U,VR3
12	VEDD VR2,VR5 → VR4=*Z/(X-Q)*
13	VEDD VR6,VR3 → VR7=*(P*Y)/U*
14	VELD L,VR8
15	VEMD VR1,VR8 → ∑
16	VEAD *,VR4 → VR5=*Z/(X-Q)+L*(W-V)*
17	VEAD VR5,VR7 → VR3=*S*
18	VESTD VR3

Курсивом в табл. 2 указаны результаты выполнения операций слева от стрелки, сохраняемые в регистрах, указанных справа от неё.

Синтаксис команды 15 означает, что в ней используется *принцип зацепления*: результат исполнения команды не будет сохранен в каком-либо векторном регистре, а будет конвейерно поэлементно передаваться напрямую в исполнительное устройство, реализующее следующую по счёту команду (здесь – сумматор для команды 16), что позволяет ещё более ускорить вычисления.

Временная диаграмма занятости исполнительных устройств и регистров векторного процессора типа 1 изображена на рис. 5 для вектора длины N=10.

Здесь MD1 – MD5 – модули 1-5 памяти данных; VR1-VR8 – векторные регистры; SUM, MUL, DIV соответственно сумматор, умножитель, делитель. Числа над стадиями команд означают номер операции из табл. 2, к которой

они относятся. Стрелками указывается применение принципа зацепления. Штриховкой отмечено смещение начало стадии L для организации зацепления.

На основании временной диаграммы рис. 5 можно вычислить время выполнения цикла на векторном процессоре типа 1 $T_{век.1}$.

$T_{век.1} = 10+N+14+N+8+N+11+N+8+N+10+N+7+N+8+N = 74+8N$ (МТ).

Необходимо отметить, что в отличие от суперскалярного, для векторных процессоров обоих типов форма зависимости $T_{век}(N)$ может изменяться при различных значениях длины вектора N (количестве итераций цикла) в связи с исчезновением конфликтов или появлением новых.

Конфликты могут быть следующими:

1) занятость модулей памяти данных;
2) отсутствие или загрузка данных в однопортовый векторный регистр, необходимых для начала текущей операции (например, при выполнении команд 5, 6, 9, 12, 15, 17, 18 рис. 5);
3) занятость векторного регистра: необходимость сохранения данных в регистр, из которого идёт чтение;
4) занятость исполнительного устройства (например, при выполнении команды 13 рис. 5).

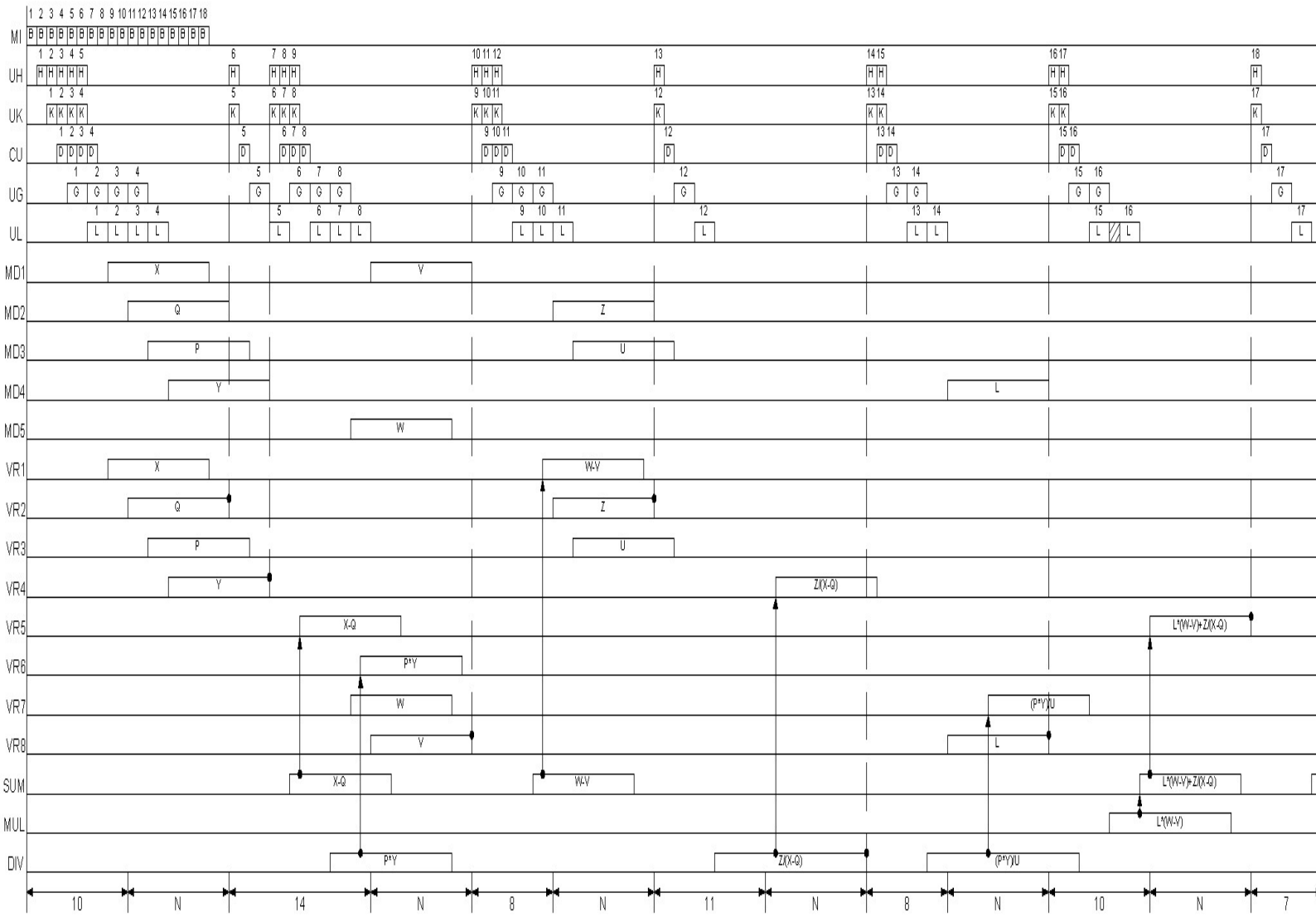


Рис 5. Временная диаграмма занятости исполнительных устройств и регистров векторного процессора с однопортовыми векторными регистрами

Пример 2. Рассмотрим теперь пример выполнения ассемблерного кода фрагмента программы **векторным процессором с двухпортовыми векторными регистрами** (векторный процессор типа 2).

Фрагмент программы следующий

$A(i) = (Z(i)+X(i))/(W(i)*P(i)) – Q(i)/Y(i) + U(i)*(V(i)+L(i)),\ i=1\ ..N$

Цель: Исследовать процесс выполнения на векторном процессоре с двухпортовыми векторными регистрами циклического фрагмента этой программы.

Пусть память команд будет единой, а память данных разделена на **5** независимых по чтению и записи модулей (**По требованию технического задания - 4 модуля памяти данных)**

Векторный процессор типа 2 имеет один конвейер команд, 8 ***двухпортовых*** векторных регистров, векторные исполнительные устройства – **2 сумматора**, 1 умножитель, 1 делитель конвейерного типа. Загрузка векторных регистров может осуществляться параллельно с выполнением арифметических операций, не связанных с загружаемым регистром.

При выполнении расчетного задания необходимо выполнить следующие действия, решив тем самым 3 основные задачи.

1. Перевести текстовое выражение фрагмента программы в ассемблерный код в соответствии с системой команд нашего векторного процессора.
2. Построить временную диаграмму выполнения векторным процессором заданного фрагмента программы.
3. Построить зависимость времени выполнения фрагмента программы от длины вектора $T_{вып} = f(N)$, при N=1÷20.

Рассмотрим подробнее каждую из этих трех задач.

Необходимо отметить, что в отличие от суперскалярного, для векторных процессоров обоих типов форма зависимости $T_{век}(N)$ может изменяться при различных значениях длины вектора N (количестве итераций цикла) в связи с исчезновением конфликтов или появлением новых.

Конфликты могут быть следующими:

1) занятость модулей памяти данных;
2) отсутствие данных в векторном регистре, необходимых для начала текущей операции;
3) занятость векторного регистра: необходимость сохранения данных в регистр, из которого идёт чтение;
4) занятость исполнительного устройства.

Необходимо запомнить простое правило использования ***двухпортовых*** векторных регистров –

из ***двухпортового*** векторного регистра можно читать, недозаписав, НО

в ***двухпортовый*** векторный регистр нельзя писать, недочитав из него.

Это простое правило определяет в, частности, что

можно использовать информацию из векторного регистра, если там есть **хотя бы один** операнд, а остальные продолжают туда дописываться;

векторный регистр **нельзя использовать для записи**, если он еще занят в другой операции.

Таким образом, наличие двух портов у векторного регистра уменьшает время выполнения фрагмента программы.

Вторым фактором, позволяющим уменьшить время выполнения программы, является наличие у векторного процессора **устройства предсказания** всех видов конфликтов, описанных выше, которое работает следующим образом:

на стадии H (стадии проверки на бесконфликтность) стандартным образом проверяется, есть ли конфликт при выполнении вычислительного процесса. Если он присутствует, то команда все равно может быть выдана в следующем такте (стадия K), в случае если конфликт разрешится **на начало стадии L** этой команды. Если конфликт присутствует, но не разрешится **на начало стадии L** этой команды, то выдача команды (стадия K) приостанавливается до момента разрешения ВСЕХ КОНФЛИКТОВ, имеющихся в этой операции.

Итак, выполним последовательно все описанные выше действия.

1. Перевести текстовое выражение фрагмента программы в ассемблерный код в соответствии с системой команд нашего векторного процессора.

Ниже, в таблице 3, приведен **один из возможных вариантов** фрагмента программы для векторного процессора типа 2. Этот ассемблерный код написан без учета особенностей функционирования векторного процессора, и, как покажет подробный анализ кода, в случае команд 4, 5 **НЕПРАВИЛЬНЫЙ**. Временная диаграмма выполнения данной программы при N=10 приведена на рис.6. (Решение задачи 2)

$$A(i) = (Z(i)+X(i))/(W(i)*P(i)) - Q(i)/Y(i) + U(i)*(V(i)+L(i)),\ i=1\ ..N$$

Таблица 3

Программа1 для векторного процессора типа 2. **НЕПРАВИЛЬНАЯ**

Номер векторной команды	**Команда**
1	VELD V, VR1
2	VELD L, VR2
3	VELD U, VR3
4	VEAD VR1, VR2 → *
5	VEMD *, VR3 → VR6
6	VELD Q, VR4
7	VELD V, VR5
8	VEDD VR4, VR5 → VR7
9	VEAD VR7, VR6 → VR8
10	VELD Z, VR1
11	VELD X, VR2
12	VELD W, VR3
13	VELD P, VR4
14	VEMD VR3, VR4 → VR5
15	VEAD VR1, VR2 → VR6
16	VEDD VR6, VR5 → VR7
17	VESD VR7, VR8 → VR1=S
18	VESTD VR1

Попробуем подробно остановиться на имеющихся ошибках.

Первая, наиболее распространенная ошибка, последствия которой надо иметь в виду сразу, при написании ассемблерного кода.

В выделенных цветом командах 4,5 используется принцип зацепления, и как было отмечено выше, они являются **единой командой**, а это значит, что **все конфликты**, которые могут быть в командах 4,5 должны проверяться **на начало стадии *L* четвертой команды**.

Команда 4,5 **не будет** иметь конфликтов, если

1) в векторных регистрах VR1, VR2 и **VR3** будет хотя бы один операнд,

2) будут свободны исполнительные устройства: сумматор и умножитель;

3) будет полностью свободен векторный регистр VR6.

Анализ временной диаграммы показывает, что **на начало стадии *L* четвертой команды** в векторных регистрах VR1, VR2 имеются операнды, сумматор и умножитель свободны; свободен векторный регистр VR6. **НО** в регистре **VR3 нет ни одного операнда**. Таким образом, **на начало стадии *L* 4** имеется один конфликт, он разрешится, когда в регистре VR3 появится хотя бы один операнд. Первый операнд появится в VR3 только через 1 такт, и так как этот конфликт единственный, именно через один такт (по условию срабатывания блока предсказания конфликтов) будет выдана стадия К следующей, четвертой команды.

Указанная ошибка на временной диаграмме рис. 6 показана кружком с минусом внутри, и отмечен момент выдачи К4.

Аналогичные конфликты возникают в командах 8, 9, 14, 16, 17,18. На диаграмме где-то они сделаны правильно, где-то неправильно. В этом случае они также показаны кружком с минусом внутри, и отмечен момент выдачи стадии К следующей команды.

Почему выше было написано, что **последствия данной ошибки** надо иметь в виду сразу, при написании ассемблерного кода. Если ассемблерный код программы останется в том виде, как он написан, но временная диаграмма будет нарисована правильно, то вся она будет состоять из сплошных разрывов на стадиях ***L***, **а оптимальность кода программы определяется минимумом разрывов во временной диаграмме,** поскольку это означает минимальность времени выполнения при заданном *N*. Если анализировать ассемблерный код вкупе с временной диаграммой, то можно сделать вывод, что если операнды использовать в следующей команде, сразу после записи в регистр, то **всегда будет разрыв** во временной диаграмме.

Вторая, грубейшая ошибка встречается реже и базируется на очень простом правиле:

Нельзя выполнять зацепление команд в случае, если длительность первой команды меньше длительности стадии *L*.

Действительно, в нашем **НЕПРАВИЛЬНОМ** случае, первая команда – сложение, длительность ее один такт, и к моменту, когда закончится стадия настройки конвейера ***L5,* из сумматора будет выдан уже второй результат**. И дальше конвейерно будет умножаться второй операнд из сумматора на первый из VR3, третий.. на второй.. и т.д. **Грубейшая ошибка**, которую можно исправить, только изменив ассемблерный код программы.

Ниже, в таблице 4, приведен следующий, также **один из возможных** вариантов фрагмента программы для векторного процессора типа 2. Этот код был написан уже с учетом особенностей функционирования векторного процессора и с учетом рассмотренных ошибок. Временная диаграмма выполнения этого кода приведена на рисунке 7. Здесь использовано 5 модулей памяти данных, (еще раз обращаю внимание, что в расчетном задании – 4 модуля памяти). **Штриховкой** показаны все интервалы занятости регистров, (**регистры заняты до окончания соответствующей операции)**. Данная штриховка сильно облегчает процесс построения временной диаграммы, поскольку четче выявляет момент окончания конфликтов.

Таблица 4

Программа 2 для векторного процессора типа 2

Номер векторной команды	**Команда**
1	VELD Q, VR1
2	VELD Y, VR2
3	VELD V, VR3
4	VEDD VR1, VR2 → VR4 (Q/Y)
5	VELD L, VR5
6	VELD U, VR6
7	VEAD VR5, VR3 → VR7 (V+L)
8	VEMD VR6, VR7 → ∑
9	VESD VR4, * → VR8
10	VELD Z, VR1
11	VELD X, VR2
12	VELD W, VR3
13	VELD P, VR4
14	VEAD VR1, VR2 → VR5
15	VEMD VR3, VR4 → ∑
16	VEDD VR5, * → /
17	VESD /, VR8 → VR6 = S
18	VESTD VR6

Рис.2.1. Временная диаграмма занятости устройств и регистров для векторного процессора с двухпортовыми реги

(со множеством ошибок)

Рис 6. Временная диаграмма **с ошибками** для случая **неправильного** кода программы

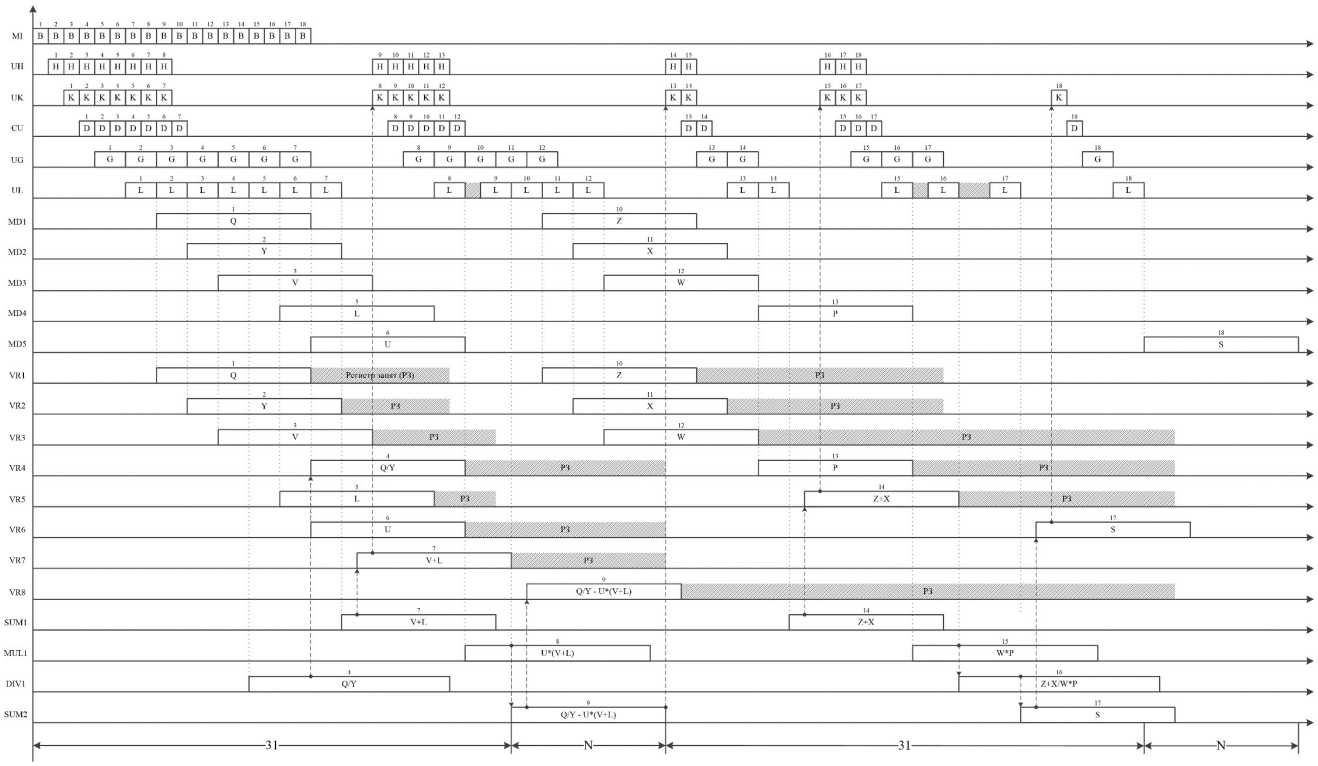


Рис 7. Временная диаграмма выполнения программы 2 при *N*=10

Таким образом, выше рассмотрены два варианта ассемблерного кода заданной программы и соответствующие им временные диаграммы, то есть выполнение **двух задач.**

Третья задача – определение зависимости времени выполнения программы от длины вектора, решается путем **анализа** построенной временной диаграммы (ВД) **для *N*=10.**

Проанализируем временную диаграмму рис 2.2 **для *N*=10** и попытаемся понять, какие интервалы на диаграмме являются постоянными, константными, а какие могут меняться в зависимости от *N*, проставляя эти интервалы **внизу диаграммы.** На имеющейся ВД это интервалы 31, *N*, 31, *N*. Таким образом,

для *N*=10 $T_{\text{вып}} = f(N) = 62 + 2N$, или в общем виде $T_{\text{вып}} = const + 2N$, где *const* определяется путем подсчета тактов на ВД.

При «чтении» диаграммы проще начать с варьируемых интервалов, зависящих от длины вектора.

1). На диаграмме под последней 18-й операцией (выгрузка результатов в память) поставлена длина интервала *N*, указывающая, что при изменении *N* длительность этой операции будет меняться. Действительно, если уменьшить *N* на 1, интервал уменьшится на 1, при увеличении *N* на 1 или 2 или… интервал будет увеличиваться на 1 или 2 или…, т.е. будет меняться в зависимости от *N*. Как

определяется местоположение этого интервала? **От начала выполнения команды** 18 (то есть, после стадии L_{18} откладывается длина вектора *N*.

2). Второй интервал *N* проставлен перед стадией K_{13}. Действительно, если уменьшить *N* на 1, интервал уменьшится на 1, при увеличении *N* на 1 или 2 или… интервал будет увеличиваться на 1 или 2 или…, т.е. будет меняться в зависимости от *N*. Как определяется? Сначала нужно определить, от чего зависит K_{13}? Сдвиг K_{13} определяется конфликтом из-за занятости регистра *VR*4, занятого в операции 8,9. При изменении длины вектора *VR*4 момент освобождения регистра сдвигается в одну или другую сторону, вправо или влево. Сам же интервал *N* проставляется влево **от конца завершения конфликта**, то есть влево от $K_{13.}$ Причем, если последовательно уменьшать *N*, то в какой-то момент, при определенном значении *N,* конец стадии L_{12} соединится с началом стадии L_{13} (в дальнейшем я назову это термином «слияние»). Это означает, что дальнейшее уменьшение *N* не приведет к изменению диаграммы, исчезнет варьируемый интервал. Данное значение *N* называется **точкой изменения зависимости $T_{вып} = f(N)$**. Зависимость времени выполнения от длины вектора изменится: $T_{вып} = const + N$. **Точкой изменения зависимости** в нашем случае будет *N*=6, при $N \leq 6$ разрыва во временной диаграмме не будет, при $N > 6$ появится разрыв и изменится зависимость $T_{вып} = f(N)$.

В общем случае, предельная точка **изменения зависимости** при $N < 10$ будет при максимально возможном **«слиянии» стадий *L*.** Например, на диаграмме рис 2.2 при $N \leq 6$ **исчезнет разрыв** между стадиями L_{12} и L_{13}, однако разрывы перед 8-ой, 15-ой и 18-ой командами останутся, поскольку они являются константными, независящими от длины вектора, т.к. при любой длине вектора на начало стадий *L* этих команд нет операнда в регистрах.

При $N > 10$ точки изменения зависимости $T_{вып} = f(N)$ определяют возможность появления новых разрывов во временной диаграмме, и, таким образом, сами эти точки определяются как раз появлением новых разрывов в ВД при увеличении *N,* то есть возможным появлением конфликтов при увеличении *N*.

В общем случае при $N > 10$ **для каждой команды** анализируется возможность **появления новых конфликтов**. Например, на диаграмме рис 2.2 при $N \geq 15$ появится новый разрыв перед 10-й командой из-за занятости регистра *VR*1, который освободится по завершении четвертой команды, в которой *VR*1 занят в предыдущей операции. Наверное, это не единственный конфликт при $N > 10$, здесь я определила лишь **принцип появления конфликтов** при увеличении *N* **относительно рассматриваемого значения**.

В этом смысле строить и анализировать **начальную** временную диаграмму можно для любого начального значения *N*, проводя дальнейший анализ в сторону уменьшения *N*, на предмет удаления всех возможных разрывов в ВД, и увеличения *N* на предмет возникновения новых разрывов в ВД. В расчетном задании

начальное значение N принято равным N =10. Именно эта временная диаграмма является отправной при проверке РЗ.

Подробный анализ построенной временной диаграммы для N=10 позволит определить все предельные точки, в которых меняется зависимость $T_{вып} = f(N)$, при N=1÷20. Однако при выполнении РЗ необходимо построить диаграммы только **для двух предельных точек**: для самого малого N и самого большого N в интервале N=1÷20.

Сравнивая зависимости $T_{с/скал.}(N)$, $T_{век.1}(N)$, $T_{век.2}(N)$ от числа итераций цикла, можно найти значения N, при которых выгодно и оправдано применение того или иного типа процессора.

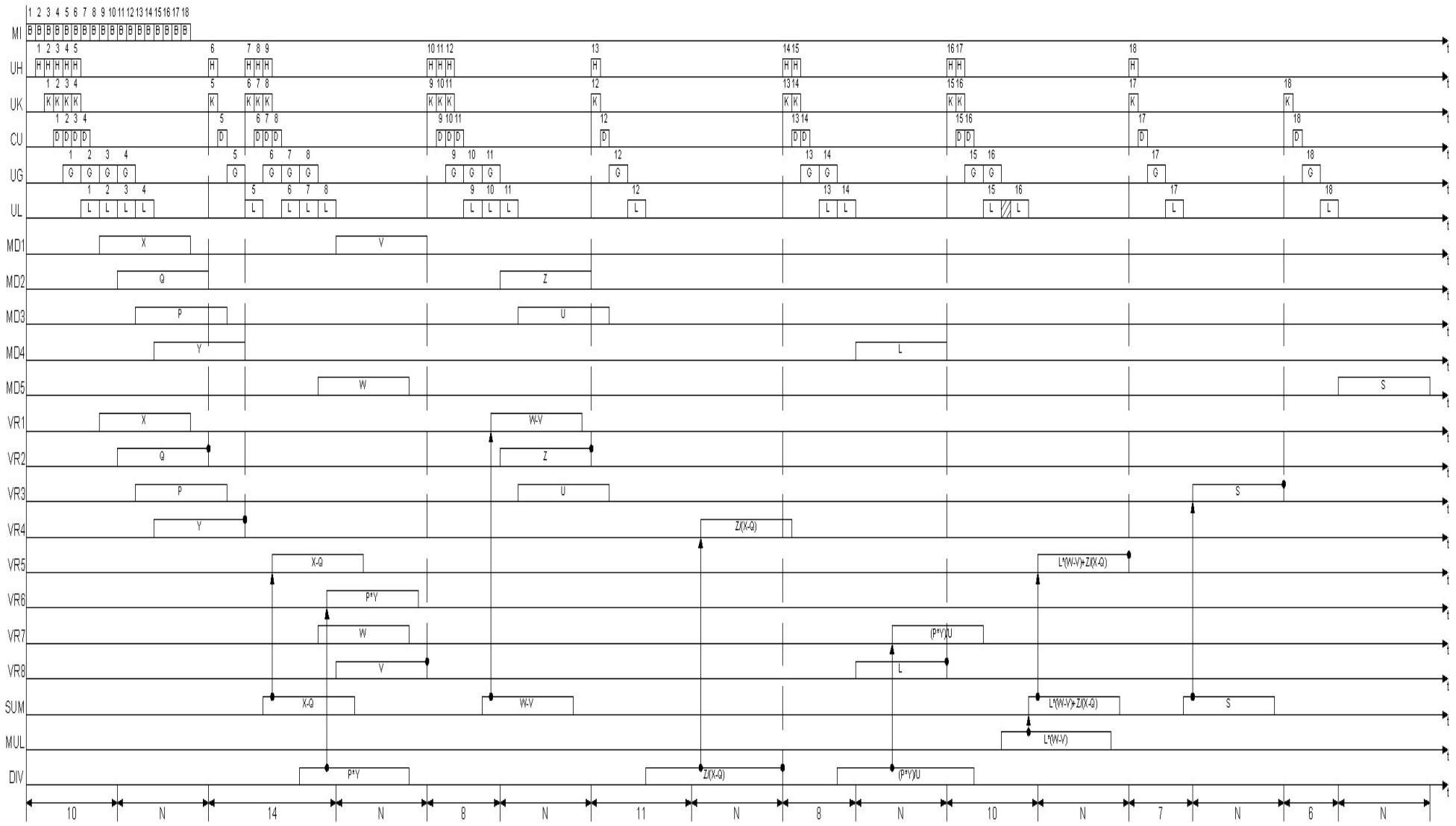


Рис. 5. Временная диаграмма занятости исполнительных устройств и регистров векторного процессора с однопортовыми регистра