

Тема 9. Скалярные и векторные процессоры

1. Введение	1
2. Две части программы – скалярная и векторная.....	3
3. Принципы скалярной обработки.....	4
4. Принципы векторной обработки.....	8
5. Дополнительные затраты на организацию векторных вычислений во время работы программы.....	15

1. Введение

Современные высокопроизводительные вычислительные системы в том числе супер-ЭВМ, имеют в своем составе несколько процессоров, чаще всего **двух типов**:

- **Скалярные**
- **Векторные**

Это связано с тем, что практически любую сложную задачу можно представить в виде совокупности двух частей: последовательной и параллельной.

Последовательные части – совокупность однократно выполняемых разнотипных операций

Параллельная часть – блоки, выполняемые параллельно. Может быть явный параллелизм, если это независимые ветви программы, или обработка потоков данных, когда над многими элементами этих потоков выполняются одни и те же операции (обычно это циклы)

Реализация циклов на высокопроизводительных **скалярных процессорах** сопровождается рядом факторов, ограничивающих максимальную скорость вычислений:

1. Перед каждой скалярной операцией необходимо вызвать и декодировать скалярную команду.
2. Для каждой команды необходимо вычислить адреса данных
3. Данные должны вызываться из памяти, а результаты запоминаться в памяти. В условиях, когда каждая команда вырабатывает свой собственный запрос к памяти, возможны конфликты при обращении к памяти
4. Реализация команд построения циклов (счетчик, переход) сопровождается накладными расходами

Ограничить влияние факторов 1-3: применить конвейер команд и отдельные буферы команд и данных

Чтобы снять влияние всех факторов на производительность ВС – используют векторные процессоры

Суперскалярный процессор представляет собой нечто большее, чем обычный последовательный (**скалярный**) процессор. В отличие от последнего, он может выполнять несколько операций за один такт. Основными компонентами суперскалярного процессора являются устройства для интерпретации команд, снабженные логикой, позволяющей определить, являются ли команды независимыми, и достаточное число исполняющих устройств. В исполняющих устройствах могут быть конвейеры. **Суперскалярные процессоры реализуют параллелизм на уровне команд.**

Примером компьютера с суперскалярным процессором является IBM RISC/6000. Тактовая частота процессора у ЭВМ была 62.5 МГц, а быстродействие системы на вычислительных тестах достигало 104 Mflop (Mflop - единица измерения быстродействия процессора - миллион операций с плавающей точкой в секунду). Суперскалярный процессор не требует специальных векторизирующих компиляторов, хотя компилятор должен в этом случае учитывать особенности архитектуры

Векторный процессор "умеет" обрабатывать одной командой не одно единственное значение, а сразу массив (вектор) значений. Пусть A_1 , A_2 и P - это три массива, имеющие одинаковую размерность и одинаковую длину, и имеется оператор $P=A_1+A_2$

Векторный процессор за один цикл выполнения команды выполнит попарное сложение элементов массивов A_1 и A_2 и присвоит полученные значения соответствующим элементам массива P . Каждый операнд при этом хранится в особом, векторном регистре. Обычному, последовательному процессору пришлось бы несколько раз выполнять операцию сложения элементов двух массивов. Векторный процессор выполняет лишь одну команду! Разумеется, реализация такой команды будет более сложной.

Векторные компьютеры различаются тем, как операнды передаются командам процессора. Здесь можно выделить следующие основные схемы:

из памяти в память - в этом случае операнды извлекаются из оперативной памяти, загружаются в арифметическое устройство и результат возвращается в оперативную память;

из регистра в регистр - операнды вначале загружаются в векторные регистры, затем операнд передается в арифметическое устройство и результат возвращается в один из векторных регистров.

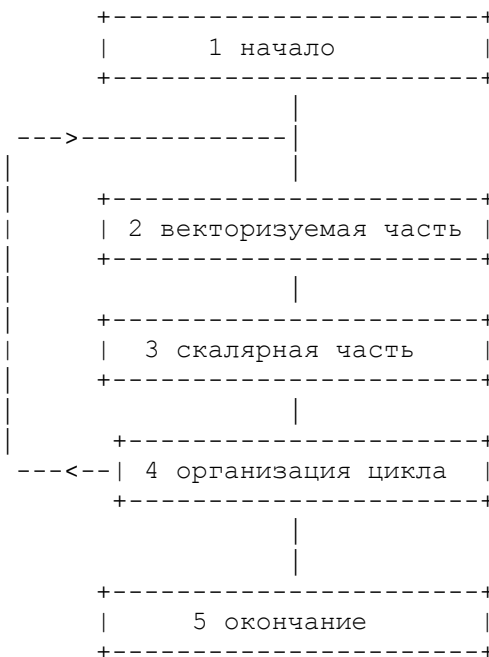
Преимущество **первой схемы** заключается в том, что она дает возможность работать с векторами произвольной длины, тогда как во втором случае требуется разбиение длинных векторов на части, длина которых соответствует возможностям векторного регистра. С другой стороны, в первом случае имеется определенное *время запуска* между инициализацией команды и появлением в конвейере первого результата. Если конвейер уже загружен, результат на его выходе будет получаться в каждом такте.

Примером ЭВМ с такой архитектурой являются компьютеры серии CYBER 200, время запуска у которых составляло до 100 тактов. Это очень большая величина, даже при работе с векторами длиной 100 элементов, вышеупомянутые компьютеры достигали лишь половины от своей максимально возможной производительности, и это их в конце концов и погубило.

В векторных компьютерах, работающих по **схеме регистр-регистр**, длина вектора гораздо меньше. Для компьютеров серии Cray это 64. Но существенно меньшее время запуска позволяет добиться хороших показателей по быстродействию. Правда, если работать с длинными векторами, их приходится разбивать на части меньшей длины, что снижает быстродействие. Векторные компьютеры, работающие по схеме из регистра в регистр, в настоящее время доминируют на рынке векторных компьютеров и наиболее наиболее известными представителями являются машины семейства Cray, NEC, Fujitsu и Hitachi.

2. Две части программы – скалярная и векторная

В любой программе существуют две части - векторизуемая и не векторизуемая. Например алгоритмы построения последовательностей, заданных рекуррентным отношением, нельзя векторизовать - каждый последующий элемент зависит от предыдущих и, соответственно, не может быть вычислен ни ранее, ни одновременно с предыдущими. Другие не векторизуемые части программ - ввод/вывод, вызов подпрограмм или функций, организация циклов, разветвленные алгоритмы, работа со скалярными величинами. Это довольно широкий класс подзадач, он гораздо шире класса векторизуемых алгоритмов. При векторизации программ на самом деле ускоряется выполнение только части программы (большей или меньшей). Поэтому каждую программу можно представить такой упрощенной диаграммой (если собрать все векторизуемые и все скалярные части в единые блоки):



Для начала исполним программу в полностью скалярном варианте (т.е. умышленно не будем векторизовать исполняемый код). Обозначим время исполнения каждой из частей программы через $T_1...T_5$. Тогда полное время работы не векторизуемых частей программы будет равно

$$T_{\text{скал}} = T_1 + T_3 + T_4 + T_5$$

а полное время работы программы будет

$$T' = T_{\text{скал}} + T_2$$

Далее перетранслируем программу в режиме векторизации. Единственная часть программы, которая ускорит свое выполнение, будет 2-ая. Предположим, что мы достигли ускорения работы этой части в N раз. Тогда полное время работы всей программы составит

$$T'' = T_{\text{скал}} + T_2/N$$

А выигрыш в эффективности работы всей программы будет

$$P = \frac{T'}{T''} = \frac{T_{\text{скал}} + T_2}{T_{\text{скал}} + T_2/N},$$

что совсем не равно N.

Положим, что $T_{\text{скал}} = 1\text{с}$, $T_2 = 99\text{с}$, а $N=100$ (очень хороший показатель для 128-элементных векторных процессоров). В нашем варианте эффективность P будет всего $(1+99)/(1+99/100)=50$ или 40% от предельно возможных 128 раз, а при $T_{\text{скал}} = 2\text{с}$ значение P будет $(2+98)/(2+98/100)=34$ (27%). Хотя само по себе увеличение быстродействия программы в 50 или даже в 30 раз при ее векторизации является очень большим (например *вычисления будут занимать 1 сутки вместо 1 месяца*), но предельно возможное ускорение в 128 (или 256) раз не может быть достигнуто на векторных ЭВМ. Практика показывает, что хорошим показателем увеличения быстродействия P можно считать уже значения 6-10, что соответствует времени исполнения скалярной части всего 10-15% от полного времени исполнения программы (T_2 составляет 85-90%).

3. Принципы скалярной обработки

Одним из самых простых и наиболее распространенных способов повышения быстродействия процессоров является конвейеризация процесса вычислений. ***Большим преимуществом конвейерных ЭВМ перед параллельными ЭВМ других типов является возможность использования пакетов программ, уже написанных для последовательных ЭВМ.***

Рассмотрим структуру и функционирование *скалярного конвейерного процессора* в целом (рис. 1).

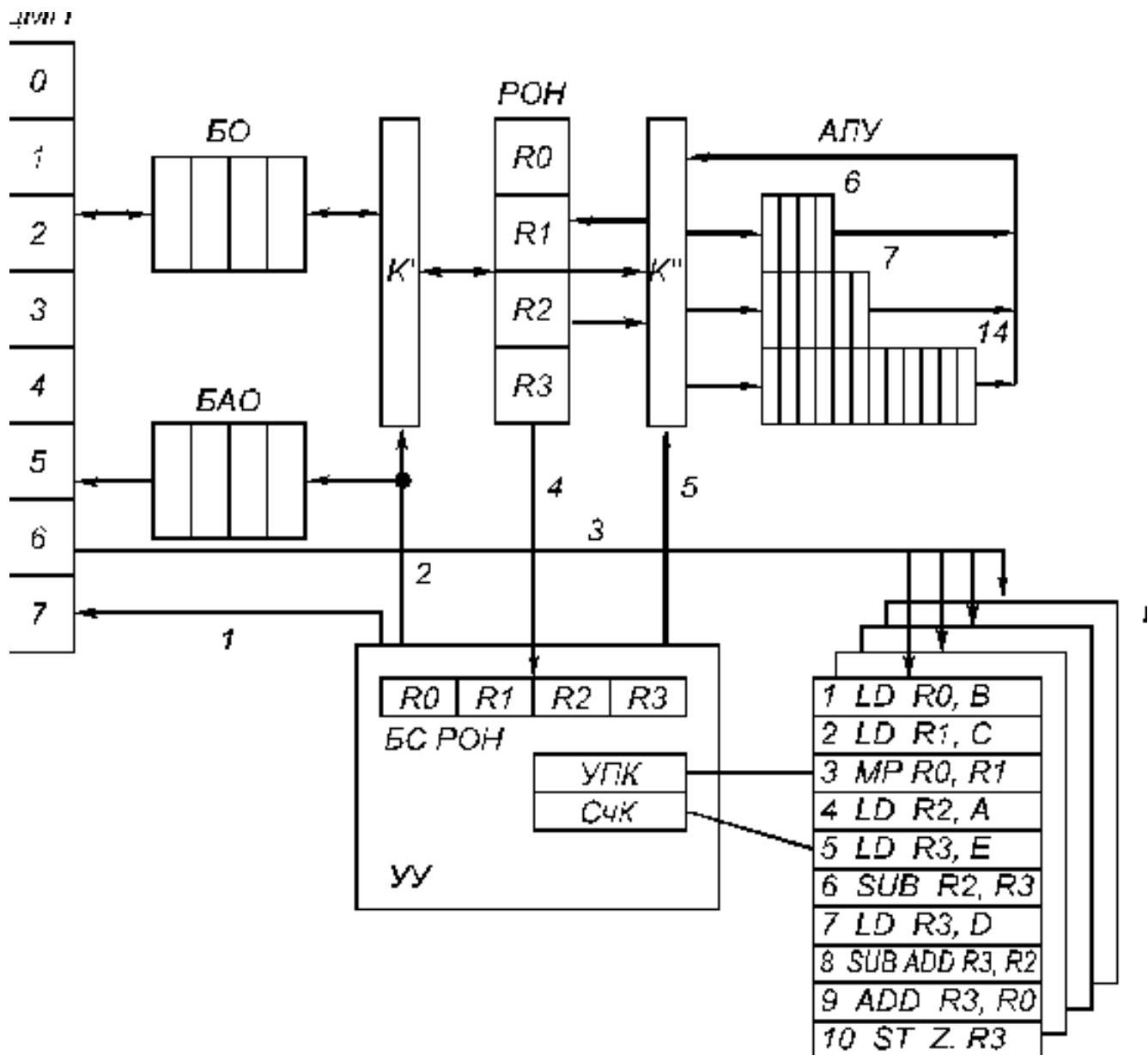


Рис. 1. Организация скалярного конвейерного процессора:

ЦМП - центральная многоблочная память;

БАО - буфер адресов операндов;

АЛУ - конвейеризованные арифметико-логические устройства для сложения, умножения и деления чисел с плавающей запятой;

К', К''- коммутаторы памяти и АЛУ соответственно;

БС РОН - блок состояния РОН;

УПК - указатель номера пропущенной команды;

СЧК - счетчик команд;

1 - шина адреса команд; 2 - шина управления выполнением команд обращения к памяти; 3 - шина заполнения БК; 4 - шина смены состояний РОН; 5 - шина управления выполнением регистровых команд

Процессор содержит несколько конвейерных АЛУ. Это позволяет одновременно исполнять смежные арифметико-логические операции, что соответствует реализации не только параллелизма служебных операций, но и локального параллелизма. Для разных операций АЛУ имеют различную длину конвейера (на рис. 1 она равна 6, 7 и 14

позициям). В процессоре используются команды двух классов: команды обращения в память и регистровые команды для работы с РОН. Буфер команд имеет многостраничную структуру, что позволяет во время работы УУ с одной страницей производить заранее смену других страниц.

Рассмотрим отрезок программы, соответствующий вычислению выражения

$$Z = A - (B * C + D) - E.$$

В программе: *LD* - команда загрузки операнда из памяти в регистр; *MP*, *SUB*, *ADD* - команды умножения, вычитания и сложения соответственно; *ST* - команда записи операнда из регистра в память.

Состояние некоторых регистров при выполнении программы показано в табл. 1. В последней колонке таблицы приведен порядок запуска команд на исполнение. В частности, видно, что некоторые команды могут опережать по запуску команды, находящиеся в программе выше запущенной. Например, команды 4 и 5 выполняются ранее команды 3. Это возможно благодаря наличию в программе локального параллелизма и нескольких АЛУ в структуре процессора. Однако подобные "обгоны" не должны нарушать логики исполнения программы, задаваемой ее информационным графом (рис. 2.). Любая операция согласно рисунку может быть запущена только после того, как подготовлены соответствующие операнды. Это достигается путем запрета доступа в определенные РОН до окончания операции, в которой участвуют данные РОН. Состояния РОН отражены в специальном БС РОН.

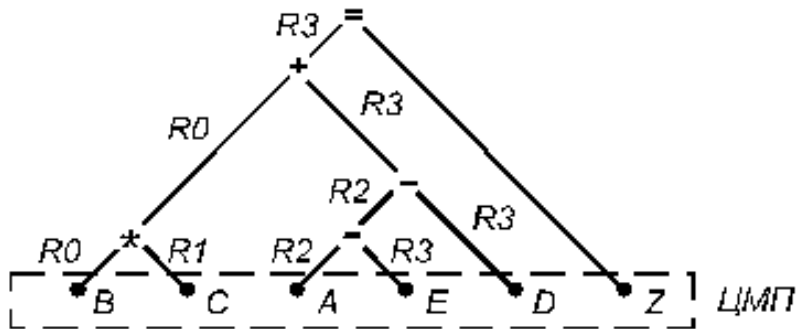


Рис. 2. Информационный граф программы: Ri - регистр общего назначения

В табл.1 приведено также описание нескольких тактов работы процессора. Принято, что выборка операнда из ЦМП занимает четыре такта. Кроме того, считается, что за один такт процессора устройство управления запускает на исполнение одну команду или просматривает в программе до четырех команд.

Таблица 1.

Порядок исполнения программы в скалярном конвейере

NN такта	Состояние РОН				Номер команды
	R0	R1	R2	R3	
1	4	-	-	-	1
2	3	4	-	-	2

3	2	3	4	-	4
4	1	2	3	4	5
5	x	1	2	3	-
6	7	-	1	2	3
7	6	-	x	1	-
8	5	-	6	x	6
9	4	-	5	4	7
10	3	-	4	3	-

Пояснения к таблице.

Такт 1. Анализ БС РОН показывает, что все РОН свободны, поэтому команда 1 запускается для исполнения в ЦМП. В столбец R0 записывается 4, что означает: R0 будет занят четыре такта. После исполнения каждого такта эта величина уменьшается на единицу. В структуре процессора занятость R0 описывается установкой разряда R0 БС РОН в 1, а затем сброс R0 в 0 по сигналу с шины 4, который появляется в такте 4 после получения операнда из памяти.

Такт 2. Запускается команда 2 и блокируется регистр R1.

Такт 3. Просматривается команда 3, она не может быть выполнена, так как после анализа БС РОН нужные для ее исполнения регистры R0 и R1 заблокированы. Команда 3 пропускается, а ее номер записывается в УПК. Производится анализ условий запуска следующей (по состоянию СчК) команды. Команда 4 может быть запущена и запускается.

Такт 4. Просмотр БК начинается с номера команды, записанной в УПК. Команда 3 не может быть запущена, поэтому запускается команда 5.

Такт 5. Команда 3 не может быть запущена, так как занят регистр R1, однако регистр R0 освободился и будет использоваться командой 3, он снова блокируется (символ x). Просмотр четырех следующих команд показывает, что они не могут быть запущены, поэтому в такте 5 для исполнения выбирается новая команда.

Такт 6. Запускается команда 3.

В дальнейшем процесс происходит аналогично. Можно заметить, что за 10 тактов, описанных в табл. 1, в процессоре запущено семь команд, что соответствует $10/7 = 1,5$ такта на команду. Предположим, что такт процессора равен 10 нс. Тогда на выполнение одной команды тратится 15 нс, что соответствует быстродействию $V = 70$ млн оп/с.

4. Принципы векторной обработки

Будем считать, что вектор - это одномерный массив, который образуется из многомерного массива, если один из индексов не фиксирован и пробегает все значения в диапазоне его изменения. В некоторых задачах векторная форма параллелизма представлена естественным образом. В частности, рассмотрим задачу перемножения матриц.

Для перемножения матриц на последовательных ЭВМ неизменно применяется гнездо из трех циклов:

```
DO 1 I = 1, L
DO 1 J = 1, L
DO 1 K = 1, L
1 C(I, J) = C(I, J) + A(I, K) * B(K, J)
```

Внутренний цикл может быть записан в виде отрезка программы на фортраноподобном параллельном языке:

```
R (*) = A (I, *) * B (*, J)      (*)
C (I, J) = SUM R (*)
```

Здесь $R(*)$, $A(I, *)$, $B(*, J)$ - векторы размерности L ;
первый оператор представляет бинарную операцию над векторами,
а второй - унарную операцию SUM суммирования элементов вектора.

Для выполнения операций над векторами также используются арифметические конвейеры. Структура конвейерного процессора для обработки векторов показана на рис.3.

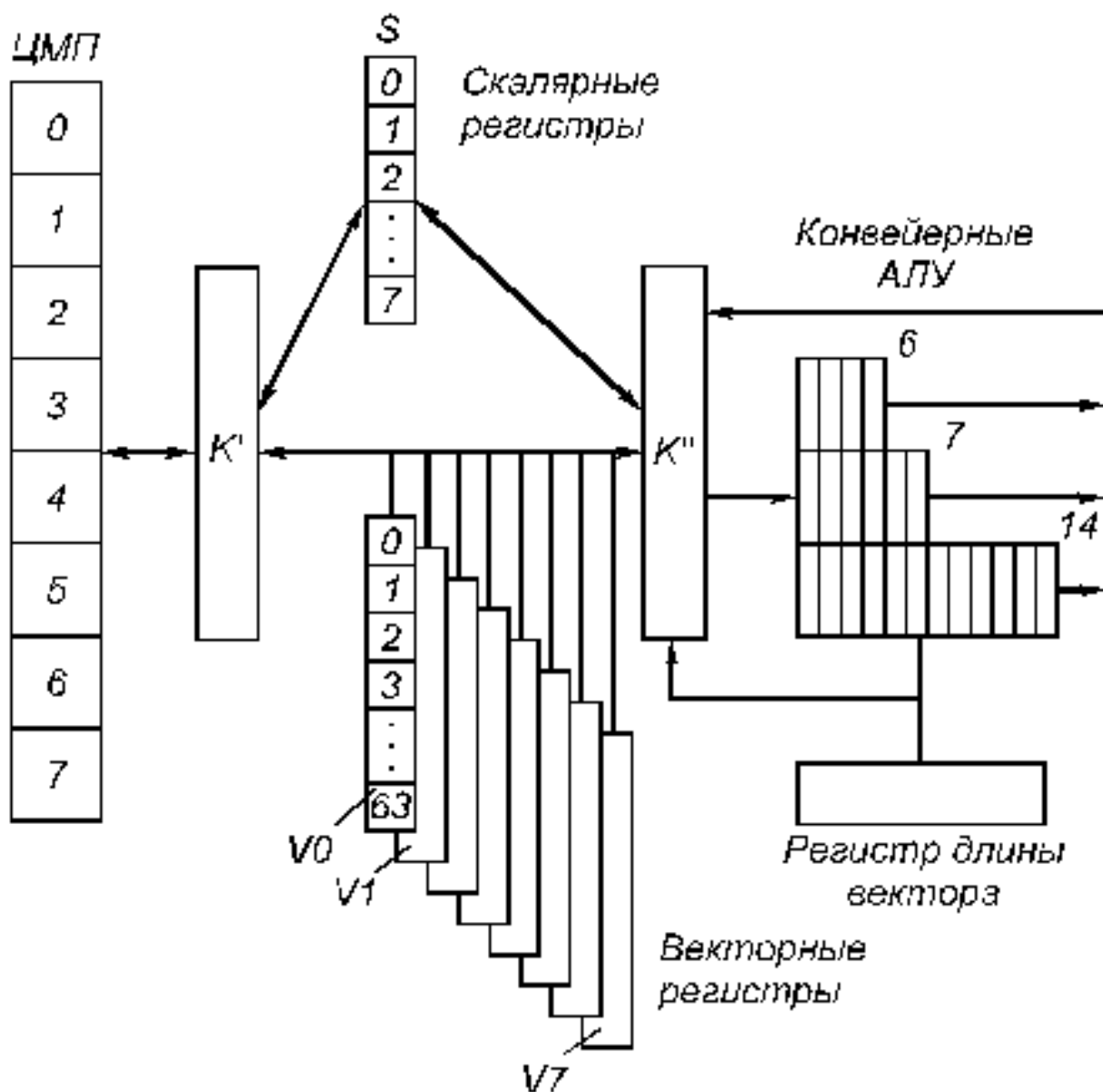


Рис. 3. Структура векторного конвейерного процессора

Структура устройства управления этого процессора не рассматривается, так как она аналогична структуре управления скалярного конвейерного процессора (отличие в том, что при выполнении векторной команды код операции команды не меняется).

Главная особенность векторного процессора - наличие ряда *векторных регистров* *V* (обычно до 8), каждый из которых позволяет хранить вектор длиной до 64 слов. Это своего рода РОН, значительно ускоряющие работу векторного процессора. Назначение остальных узлов такое же, как и узлов, изображенных на рис. 2.

Рассмотрим, как будет выполняться программа (*) в векторном процессоре. На условном языке ассемблерного уровня программа может быть представлена следующим образом:

```

1 LD L, Vi, A
2 LD L, Vj, B
3 MP Vk, Vi, Vj      (**)
4 SUM Sn, Vk

```

Операторы 1 и 2 соответствуют загрузке слов из памяти с начальными адресами A и B в регистры V_i , V_j ; оператор 3 означает поэлементное умножение векторов с размещением результата в регистре V_k ; оператор 4 - суммирование вектора из V_k с размещением результата в S_n .

Соответственно этой программе векторные регистры сначала потактно заполняются из ЦМП, а затем слова из векторных регистров потактно (одна пара слов за такт) передаются в конвейерные АЛУ, где за каждый такт получается один результат.

Рассмотрим характеристики быстродействия векторного процессора на примере выполнения команды $MP\ V_i, V_j, V_k$. Число тактов, необходимое для выполнения команды, равно: $r = m_* + L$, где m_* - длина конвейера умножения. Поскольку на умножение пары операндов затрачивается $k = (m_* + L)/L$ тактов, то быстродействие такого процессора

$$V = 1 / (K \Delta t) = \frac{L}{(m_* + L) \Delta t},$$

где Δt - время одного такта работы конвейера.

Быстродействие конвейера зависит от величины L (рис. 4, кривая 1).

При $L > m_*$ величина $K \sim 1$ и $V \sim 1/\Delta t$. Обычно для векторных процессоров стараются сделать Δt малым, в пределах 10...20 нс, поэтому быстродействие при выполнении векторных операций может достигать 50...10 млн оп/с.

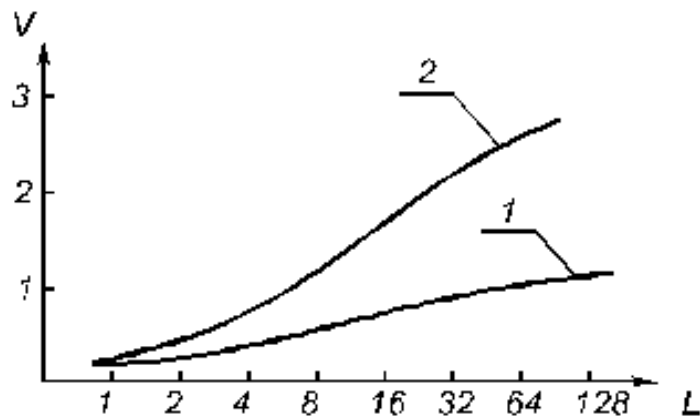


Рис.4. Зависимость быстродействия векторного процессора от длины вектора: $m_{\text{ЦМП}} = 4$; $m_* = 7$; $m_+ = 6$

Важной особенностью векторных конвейерных процессоров, используемой для ускорения вычислений, является механизм *зацепления*. Зацепление - такой способ вычислений, когда одновременно над векторами выполняется несколько операций. В частности, в программе можно одновременно производить выборку вектора из ЦМП, умножение векторов, суммирование элементов вектора. Поэтому программу можно переписать следующим образом:

$LD\ L, V_i, A$
 $3C\ S_n, V_i, B$

Здесь команда зацепления (ЗЦ) задает одновременное выполнение операций в соответствии со схемой соединений (рис.5).

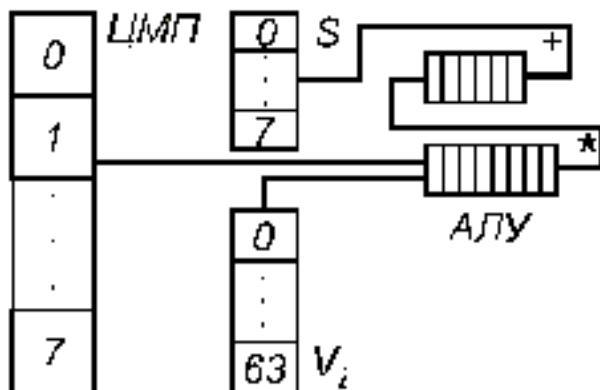


Рис. 5. Выполнение операции зацепления в векторном конвейерном процессоре

Для команды ЗЦ получаем:

$$r = m_{\text{ЦМП}} + m_* + m_+ + L,$$

$$K = (m_{\text{ЦМП}} + m_* + m_+ + L) / L,$$

$$V = n / (K \Delta t),$$

где n - число одновременно выполняемых операций. В случае команды ЗЦ для $n = 3$ быстродействие процессора возрастает (рис. 4, кривая 2).

При $L \gg m_i$ и $\Delta t = 10 \dots 20$ нс в зацеплении быстродействие равно $150 \dots 300$ млн оп/с. Такое быстродействие достигается не на всех векторных операциях. Для векторных ЭВМ существуют "неудобные" операции, в которых ход дальнейших вычислений определяется в зависимости от результата каждой очередной элементарной операции над одним или парой операндов. В подобных случаях L приближается к единице. **К таким операциям относятся операции рассылки и сбора**, которые можно определить следующими отрезками фортран-программ:

1) рассылка

```
DO 1 I = 1, L
1 X(INDEX (I)) = Y (I)
```

2) сбор

```
DO 1 I = 1, L
1 Y (I) = X (INDEX (I))
```

Целочисленный массив INDEX содержит адреса операндов, разбросанных произвольным образом в памяти процессоров. Операция рассылки распределяет упорядоченный набор элементов $Y(I)$ по всей памяти в соответствии с комбинацией адресов в массиве INDEX. Операция сбора, наоборот, собирает разбросанные элементы X

в упорядоченный массив Y . Названные операции имеются в задачах сортировки, быстрого преобразования Фурье, при обработке графов, представленных в форме списка, и во многих других задачах.

Быстродействие векторного процессора на таких операциях снижается до уровня быстродействия скалярного процессора.

Выше конвейерные ЭВМ были описаны по частям: сначала скалярная часть, затем векторная.

Теперь рассмотрим полную структуру векторной ЭВМ на примере машины CRAY (рис. 6). Архитектура типичного векторного суперкомпьютера Cray следующая (в качестве примера рассмотрим старую модель Cray-1):

- Процессор имеет 8 векторных регистров, каждый из которых может хранить 64 слова по 64 бита каждое.
- Есть также 8 скалярных регистров для 64-битовых слов и
- 8 регистров для адресации для 20-битовых слов.
- Вместо кэш-памяти используются специальные регистры, управление которыми осуществляется программным путем из выполняющейся программы.

Всего компьютер Cray-1 содержал до 12 конвейеризованных процессоров, имевших отдельные конвейеры для различных арифметических и логических операций. Скорость работы процессора строго согласована со скоростью работы оперативной памяти.

CRAY-1, созданная в 1976 г., была одной из первых ЭВМ, в которой в полной мере проявились все характерные особенности *векторно-конвейерных машин*. Машина CRAY-1 включает стандартные для любой ЭВМ секции: управления памятью и ввода-вывода, регистровую секцию и группу функциональных устройств. Однако состав оборудования каждой секции существенно отличается от аналогичных устройств последовательных ЭВМ.

Память используется как для выборки команд и векторных данных (конвейерный режим), так и для выборки скалярных данных (поточный режим). Для организации поточного режима применяются буферные регистры адреса ($V0...V63$) и данных ($T0...T63$). Память обладает большой пропускной способностью, однако для разных узлов не всегда используется полная пропускная способность. Максимальная скорость нужна для заполнения буферов команд.

Структура УУ: здесь имеются механизмы предварительного просмотра команд, а также резервирования регистров и функциональных устройств.

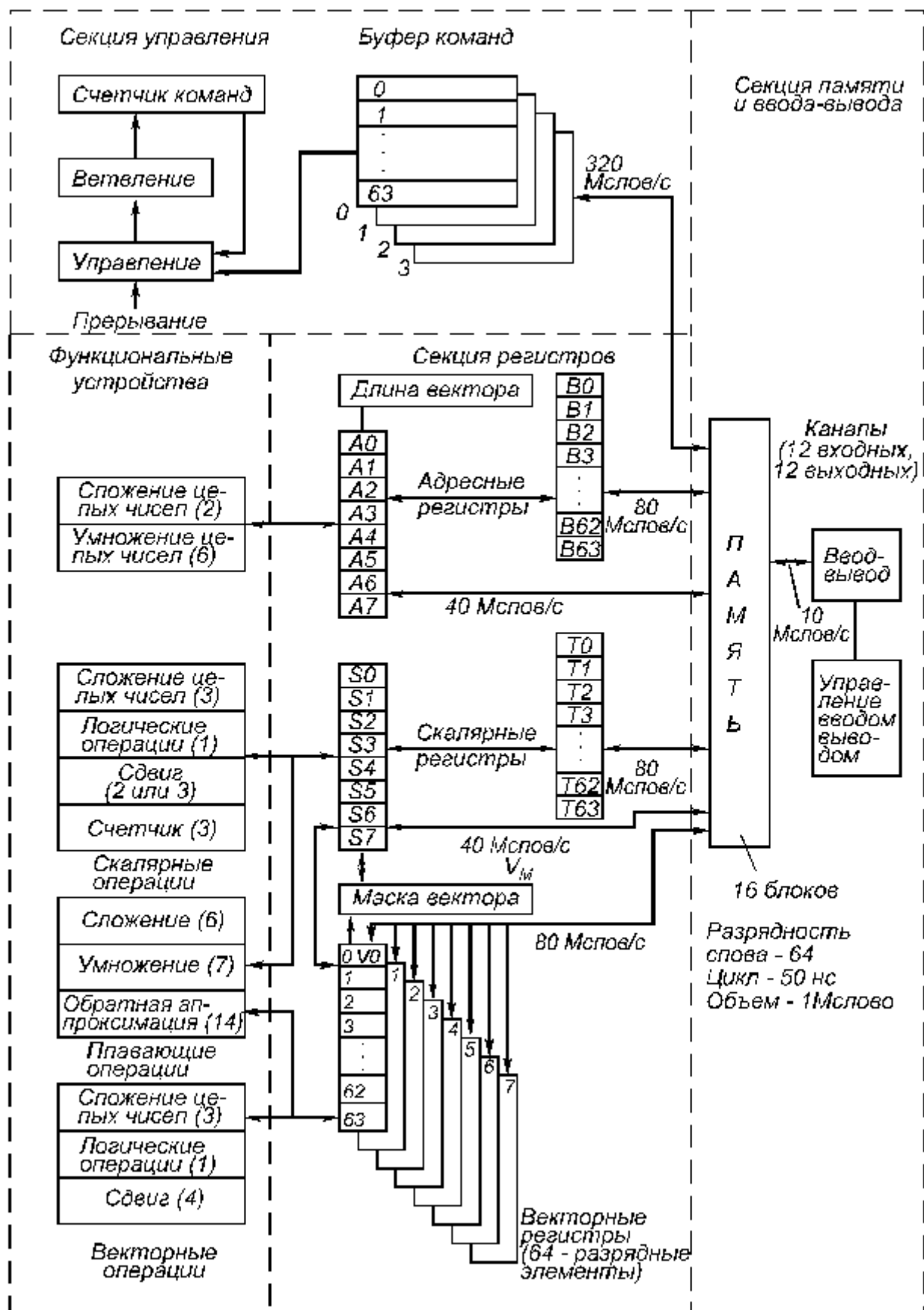


Рис. 6. Архитектура процессора CRAY-1.

Для выполнения адресных операций используются группа адресных регистров $A0...A7$ и специальные АЛУ для операций целочисленной арифметики. Эти АЛУ, как и все другие функциональные устройства, имеют конвейерную структуру.

Скалярные операции выполняются с помощью группы регистров $S0...S7$, а векторные - с помощью векторных регистров $V0...V7$.

Наибольший интерес представляет состав функциональных устройств. В выполнении скалярных операций участвует семь функциональных устройств: четыре для работы с целочисленной арифметикой и три для работы с числами с плавающей запятой. Устройство "Счетчик" используется для подсчета числа единиц в операнде или числа нулей, предшествующих первой единице операнда.

Поскольку деление плохо поддается конвейеризации, в CRAY-1 оно выполняется в устройствах обратной аппроксимации и умножения посредством итерационной процедуры. Такой подход позволяет использовать конвейерную обработку на операции умножения и зацепление операций.

Векторные операции в CRAY-1 можно разделить на четыре типа. Векторная инструкция первого типа получает операнды из одного или двух регистров V и отправляет результат в другой регистр V . Последовательные пары операндов передаются из V_j и V_k в конвейерное АЛУ в каждом такте, и соответствующий результат на выходе АЛУ появляется на m тактов позже, где m - длина конвейера. Результат отправляется в регистр результата V_i . Векторная инструкция второго типа получает по одному операнду из регистров S и V . Инструкции двух других типов передают данные между памятью и регистрами V .

При выдаче векторной инструкции требуемое АЛУ и регистры операндов резервируются на число тактов, определяемое длиной вектора. Последующая векторная инструкция, требующая тех же ресурсов, что и выполняемая, не может выполняться до тех пор, пока ресурсы не будут освобождены. Однако выполнение последующих инструкций, не пересекающихся по ресурсам с неоконченной инструкцией, разрешается.

Две команды машины CRAY-1 требуют специального объяснения. Установка маски 64-разрядного векторного регистра маски (VM) соответствует 64 элементам векторного регистра. Если элемент удовлетворяет условию, то соответствующий разряд VM устанавливается в 1, в противном случае - в 0. Таким образом команда, $VM\ V5, Z$ устанавливает разряд VM в 1, когда элементы $V5$ равны нулю; $VM\ V7, P$ устанавливает разряд VM в 1, если элементы $V7$ положительны.

По команде слияния векторов содержимое двух векторных регистров V_i и V_k сливается в один результирующий вектор V_i в соответствии с маской регистра VM . Если l -й разряд VM - единица, то l -й элемент V_j становится l -м элементом регистра результатов, в противном случае l -й элемент V_k становится l -м элементом регистра результатов. Значение регистра длины вектора определяет число сливаемых элементов. Таким образом, команда $V_iV_j!V_k&VM$ сливает V_j и V_k в V_i в соответствии с комбинацией в VM ; $V7S2!V6&VM$ сливает $S2$ и $V6$ в $V7$ в соответствии с комбинацией в VM .

Цель команд маски и слияния - обеспечить условные вычисления с векторными командами.

В CRAY-1 соотношение объема памяти хорошо сбалансировано с производительностью системы. По эмпирическому правилу Амдала 1 бит памяти должен приходиться на 1 оп/с. В CRAY-1 при памяти 1 Мслов (64 Мбит) производительность равна 80 Моп/с.

В данной системе впервые применено зацепление операций. За счет этого могут работать два, а иногда три функциональных устройства, доводя производительность системы до 160 и даже 240 миллионов результатов с плавающей запятой в секунду.

5.Дополнительные затраты на организацию векторных вычислений во время работы программы

Для работы на векторных ЭВМ наиболее удобными являются массивы с длиной, равной длине векторного регистра. Однако это благое пожелание очень редко выполняется. Более того, часто число элементов массива вообще не кратно 128. Рассмотрим простейший цикл, который можно векторизовать. Пусть дан массив m длины 128, который надо заполнить по следующему алгоритму:

```
do i=1, 128
    m(i) = i
enddo
```

Мы будем пользоваться командами ассемблера несуществующей ЭВМ, но вполне отражающими смысл операций с векторными регистрами. Приведенный цикл можно записать на ассемблере так:

```
SETLEN #128 ; установить используемое число
              ; элементов в векторных регистрах (во всех)
SETINC #4    ; установить смещение к последующему
              ; элементу массива в памяти (4 байта)
SETNUM v0    ; записать в элементы векторного регистра v0
              ; их номера начиная с нуля и кончая 127
ADD #1, v0   ; добавить 1 к каждому элементу регистра v0
SAVE v0, m   ; записать элементы v0 в ОЗУ в последовательные
              ; слова (смещение = 4) начиная с адреса m
```

Обратите внимание на 3 команду - каждый элемент векторного регистра "знает" свой номер. Это делает очень простым вычисление переменной цикла: после добавления 1 значение переменной получается записанным в соответствующий элемент вектора. Всего 5 последовательных команд векторного процессора выполняют цикл из 128 повторений. Здесь нет ни команд сравнения, ни условных переходов.

Теперь увеличим размер массива и число повторений цикла до N :

```
do i=1, N
    m(i) = i
enddo
```

Для правильной работы процессора мы обязаны установить число используемых элементов вектора не более, чем 128. Если N будет произвольным, то нам придется превратить данный одинарный цикл в двойной:

```
1    do inc=0, N-1, 128
2        NN = min0( 128, N-inc )
3        do i=1, NN
            m(inc+i) = inc+i
        enddo
    enddo
```

Цикл с меткой 1 выполняет "разбиение" массива на подмассивы длиной 128 элементов. Переменная inc имеет смысл смещения от первого элемента массива к очередному подмассиву: 0, 128, 256... Переменная NN определяет длину подмассива. Обычно она равна 128, но последний подмассив может иметь меньшую длину, если N не кратно 128. Функция $min0$ выбора минимального значения в операторе с меткой 2 выдает значение не 128 только для последнего подмассива. Внутренний цикл с меткой 3 практически эквивалентен циклу из предыдущего примера.

Имеется только 3 отличия:

- число элементов в векторном регистре равно NN ,
- к параметру цикла дополнительно надо добавлять значение inc ,

- регистр записывать в память начиная не с начала массива, а с элемента с номером $i+inc$

Этот цикл может быть записан в машинных командах примерно так:

```
SETLEN NN      ; число элементов в векторных регистрах = NN
SETINC #4      ; смещение к последующему элементу массива
SETNUM v0      ; записать в элементы векторного регистра v0
                ; их номера начиная с нуля и кончая 127
ADD #1, v0     ; добавить 1 к каждому элементу регистра v0
ADD inc, v0    ; добавить значение переменной inc
MOVE inc, r0   ; записать в скалярный регистр r0 значение
                ; переменной inc - смещение от первого
                ; элемента массива к m(inc+1)
MUL #4, r0     ; умножить на 4 - смещение в байтах
ADD #m, r0     ; добавить адрес массива m, получается адрес
                ; элемента m(inc+1)
SAVE v0, @r0   ; записать элементы v0 в ОЗУ в последовательные
                ; слова (смещение = 4) начиная с адреса,
                ; хранящегося в регистре r0
```

По отношению к простейшему циклу добавились **одна векторная и три скалярных** команды. Однако это только внутренний цикл. Охватывающий его цикл с меткой 1 и вычисление NN создадут дополнительный код, который будет выполняться столько же раз, сколько и код для внутреннего цикла. Транслятор всегда будет создавать охватывающий цикл, если N есть переменная, а не константа со значением от 1 до 128 (для 128-элементного векторного регистра).

Из сказанного выше следует, что **эффективность векторной программы будет невысокой при работе с небольшими массивами, длина которых заранее неизвестна. Циклы с тремя повторениями могут в векторном режиме выполняться медленнее, чем в скалярном на той же машине.**