Эффекты, возникающие при БПФ вследствие конечной разрядности чисел

1. Oppenheim A. V., Weinstein C. J., Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform, *Proc. IEEE*, **60**, No. 8, 957—976 (Aug. 1972); есть русский перевод: Оппенгейм, Вайнштейн, Влияние конечной длины регистра при цифровой фильтрации и быстром преобразовании Фурье, *ТИИЭР*, т. 60, № 8, стр. 41—65 (1969).
2. Welch P. D., A Fixed-Point Fast Fourier Transform Error Analysis, *IEEE Trans. Audio and Electroacoustics*, **AU-17**, No. 2, 151—157 (June 1969).
3. Weinstein C. J., Roundoff Noise in Floating Point Fast Fourier Tranform Computation, *IEEE Trans. on Audio and Electroacoustics*, **AU-17**, No. 3, 209—215 (Sept. 1969).
4. Kaneko T., Liu B., Accumulation of Round-Off Errors in Fast Fourier Transforms, *J. Assn. Comp. Mach.*, **17**, No. 4, 637—654 (Oct. 1970).

Глава 11

УНИВЕРСАЛЬНЫЕ УСТРОЙСТВА В СИСТЕМАХ ОБРАБОТКИ СИГНАЛОВ

11.1. Введение

При решении любой конкретной задачи специализированная аппаратура всегда эффективнее универсальных ЦВМ. Однако требуемая в очень многих случаях гибкость вычислительных средств проще всего обеспечивается с помощью универсальных ЦВМ. Именно таким специальным приложениям и посвящена настоящая глава. В ней рассматриваются вопросы проектирования универсальных устройств, предназначенных для исследования методов обработки сигналов при анализе речи, в радиолокации, гидролокации, сейсмологии и технике связи. Дело в том, что в перечисленных областях такие устройства оказываются эффективнее больших вычислительных машин, обслуживающих в режиме разделения времени все увеличивающееся число экспериментаторов. Основным назначением этих устройств является создание соответствующих алгоритмов путем моделирования разрабатываемых специализированных устройств.

В настоящее время вычислительная техника все еще продолжает стремительно развиваться. Целесообразность той или иной структуры вычислительной машины в значительной степени зависит от характеристик существующих компонентов. Так, например, когда основным видом запоминающих устройств ЦВМ были кубы памяти на магнитных сердечниках, полупроводники только начинали заменять радиолампы в арифметических устройствах, так что даже обыкновенный триггер был громоздким и дорогим. При этом внешняя (относительно куба) память была весьма дорогостоящей, и проектировщики старались уменьшить ее объем. Однако позднее, когда память на сердечниках все еще доминировала, но триггеры и арифметические устройства стали недорогими устройствами с большими функциональными возможностями, начали широко применяться общие регистры и сверхоперативные ЗУ. Появление микросхем со средним уровнем интеграции привело к быстрому внедрению принципов параллельной обработки, многие из которых и в настоящее время не являются бесспорными. Быстрый прогресс в области создания новых компонентов стимулировал

появление множества идей, но пока эти компоненты не нашли широкого применения.

В данной главе сначала в качестве примера рассматривается универсальная ЦВМ и обсуждаются возможности ее применения для обработки сигналов. После этого описываются различные методы увеличения скорости обработки сигналов на универсальной ЦВМ. В качестве примеров структуры универсальных ЦВМ, предназначенных для обработки сигналов, и применения их в соответствующих системах в конце главы подробно рассматриваются два быстродействующих цифровых процессора для анализа сигналов — FDP (Fast Digital Processor) и LSP2 (Lincoln Signal Processor 2).

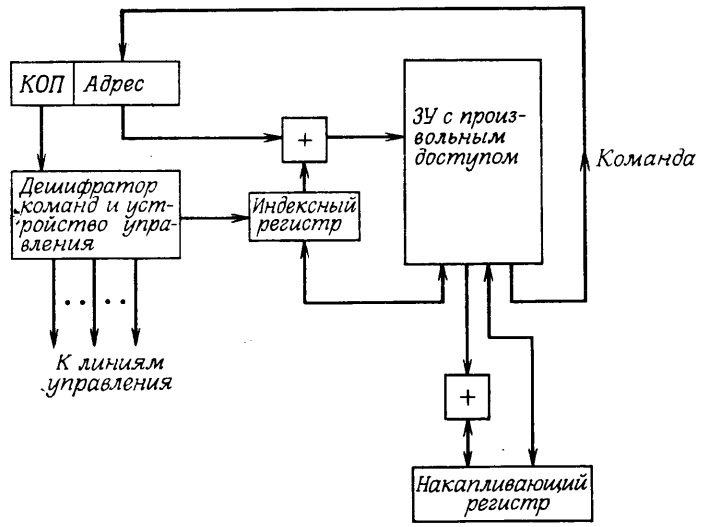
11.2. Специализированные и универсальные вычислительные машины

Реализация алгоритмов обработки сигналов цифровыми средствами предполагает использование либо универсальной, либо специализированной вычислительной машины. Различие между ними всегда было (и сейчас остается) довольно неопределенным, поэтому имеет смысл обсудить оба эти термина.

Под *универсальностью* подразумевается гибкость, программируемость, возможность решения различных задач. Эти качества необходимы весьма часто, и за них мы готовы расплачиваться деньгами, временем, габаритами, потребляемой мощностью и т.д. Однако во многих случаях оказалось лучше купить и запрограммировать серийную ЦВМ потому, что она справляется с поставленной задачей, уже выпускается промышленностью и, кроме того, дешевле, чем разработка и изготовление даже сравнительно несложного цифрового блока. Можно привести по крайней мере один пример, когда явно справедливо обратное, а именно когда требования к быстродействию столь велики, что для выполнения обработки необходимы специализированные структуры. По-видимому, наиболее очевидное различие между универсальной и специализированной вычислительными системами заключается в том, что для любого алгоритма, запрограммированного на универсальной ЦВМ, можно построить специализированное устройство, с помощью которого этот алгоритм выполняется быстрее. Действительно, при реализации заданного алгоритма в специализированном устройстве часто удается эффективно использовать методы параллельной обработки. В универсальных ЦВМ параллелизм в принципе также возможен и иногда действительно вводился, но опыт показал, что в этом случае он оказывается не столь эффективным, усложняет программирование и затрудняет взаимодействие отдельных частей ЦВМ. Нам представляется, что вообще при построении гибких ЦВМ, предназначенных для широкого круга алгоритмов, последовательная структура оказывается более подходящей. Таким образом, при проектировании самых разнообразных цифровых устройств, от вычислительных систем общего назначения или систем, ориентированных на ограниченный круг задач (например, в радиолокации), и до узкоспециализированных бортовых устройств, приходится находить наилучшее соотношение между быстродействием и гибкостью.

11.3. Способы описания вычислительных машин

Любую вычислительную машину, как специализированную, так и универсальную, можно описать, задав: 1) отдельные вычислительные блоки, такие, как сумматоры, умножители и логические цепи; 2) структуру памяти; 3) все связи между блоками памяти и вычислительными блоками; 4) последовательность операций; 5) соединения с устройствами, рассматриваемыми как *внешние* по отношению к вычислительной машине. Специализированные вычислительные машины предназначены для выполнения ограниченного числа алгоритмов, в каждом из которых используется фиксированная последовательность операций. Универсальные ЦВМ имеют программу, сохраняемую в одном или нескольких блоках памяти, причем каждая «строчка» программы представляет собой команду, непосредственно выполняемую машиной, а весь алго-



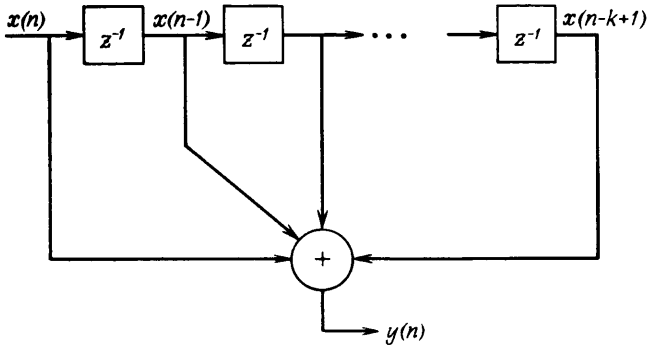
Фиг. 11.1. Структурная схема упрощенной универсальной вычислительной машины.

ритм представляется программистом в виде последовательности таких команд.

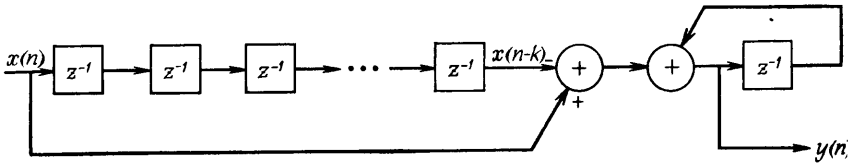
Центральным элементом упрощенной универсальной ЦВМ (фиг. 11.1) является запоминающее устройство (ЗУ) с произвольным доступом. В нем хранятся и обрабатываемые данные, и программа. В рассматриваемой упрощенной схеме команда состоит из кода операции (КОП) и адреса. В машине имеется один индексный регистр, содержимое которого можно (по желанию) добавлять к адресной части команды, формируя тем самым истинный адрес ячейки памяти. Арифметическое устройство ЦВМ состоит из сумматора и накапливающего регистра. Линии со стрелками представляют собой возможные линии связи. Число линий связи является важным фактором, определяющим стоимость ЦВМ. Для каждой из них должна быть предусмотрена схема коммутации. С помощью этих схем устройство управления подключает линии, необходимые для выполнения текущей команды. В данном разделе не рассматривается, каким образом информация (программа и данные) была введена в память. Эта операция зависит от вида системы ввода — вывода, описываемой в последующих разделах главы.

Рассмотрим некоторые типичные команды ЦВМ, а затем попробуем составить программу вычисления скользящей суммы — одного из простейших алгоритмов обработки сигналов:

Название команды	*Описание*
1. Load Y (загрузить Y)	Передать содержимое регистра памяти Y в накопитель.
2. Store Y (запомнить Y)	Передать содержимое накопителя в регистр памяти Y.
3. Add Y (сложить Y)	К величине, уже содержащейся в накопителе, добавить содержимое регистра памяти Y; результат оставить в накопителе.
4. Sub Y (вычесть Y)	То же, но выполняемым действием является вычитание.
5. JNX Y (перейти по X в Y)	Если содержимое индексного регистра X меньше нуля, то перейти к выполнению команды, хранящейся в регистре памяти Y; в противном случае перейти к следующей команде. В обоих случаях после перехода увеличить X на 1.
6. YIX Y (занести в X из Y)	Занести число из адресной части выполняемой команды в индексный регистр X.
7. Clear Y (очистить Y)	Записать нуль в регистр памяти Y.
8. Halt (стоп)	



Фиг. 11.2. Блок-схема алгоритма вычисления скользящей суммы.



Фиг. 11.3. Блок-схема рекурсивного алгоритма вычисления скользящей суммы.

11.4. Программа вычисления скользящей суммы

Скользящей суммой называют величину

$$y(n)=\sum_{m=0}^{k-1} x(n-m). \tag{11.1}$$

Для увеличения скорости вычисления скользящей суммы (при $k\geqslant 3$) целесообразно представить ее рекурсивной формулой

$$y(n)=y(n-1)+x(n)-x(n-k). \tag{11.2}$$

На фиг. 11.2 представлена схема вычисления $y(n)$ путем простого суммирования чисел на выходах цепочки цифровых элементов задержки. На фиг. 11.3 показана другая схема реализации того же алгоритма, составленная из последовательно соединенных трансверсального фильтра и однополюсного рекурсивного фильтра. В табл. 11.1 приведен формат данных, удобный для реализации алгоритма (11.2) с помощью упрощенной универсальной вычислительной машины.

Чтобы не усложнять программу дополнительными операциями, учитывающими краевые эффекты, массив данных $x(0)$ — $x(999)$ дополняется с обеих сторон нулевыми отсчетами, число которых на

Таблица 11.1

Формат входных данных для программы вычисления скользящей суммы

DATA	→	0
		0
		.
		.
		.
		0
DATA + 26	→	x (0)
		x (1)
		.
		.
		.
DATA + 1025	→	x (999)
		0
		0
		0
		.
		.
		.
DATA + 1051	→	0
		.
		.
		.
YN	→	0
		.
		.
		.

единицу меньше количества слагаемых в сумме. Здесь для определенности взято $k = 26$. Метки, расположенные слева от стрелок в табл. 11.1, обозначают адреса, используемые в приводимой ниже программе вычисления скользящей суммы:

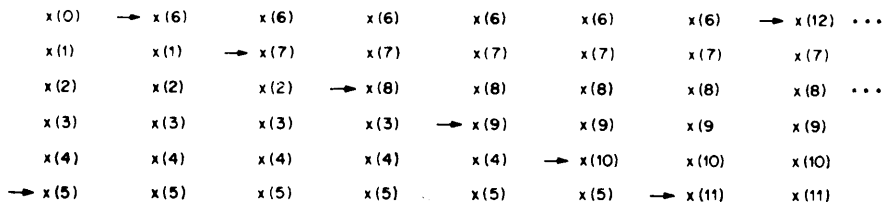
			Пояснения
	YIX	—1025	Занести число —1025 в регистр X
	CLEAR	YN	Занести нуль в регистр YN
LOOP →	ADD$_x$	DATA + 1051	Прибавить $x(n)$ к $y(n-1)$
	SUB$_x$	DATA + 1025	Сформировать в накопителе $x(n) - x(n-k) + y(n-1)$
	STORE$_x$	YN + 1025	Запомнить $y(n)$
	JNX	LOOP	Повторить в цикле 1024 раза
	HALT		Стоп

Из приведенной программы видно, что для обработки каждого входного отсчета приходится выполнять четыре команды. Естественно, что время обработки зависит от быстродействия узлов ЦВМ. Для удобства предположим, что цикл обращения к памяти равен 100 нс. Каждая из первых трех команд программного цикла требует не менее *двух* циклов обращения к памяти: одного при вызове команды, другого при ее исполнении. Таким образом, полное время, необходимое для обработки одного отсчета, равно 600 нс плюс время выполнения условного перехода (JNX) и время выполнения арифметических операций, включить которое в цикл обращения к памяти затруднительно. В итоге получается около 1 мкс на каждый входной отсчет.

11.5. Особенности ввода — вывода при обработке в реальном времени

При обработке сигналов на ЦВМ в реальном времени возникает много дополнительных трудностей. В рассматриваемом в разд. 11.4 примере обработки не в реальном времени предполагалось, что обрабатываемый массив из 1000 отсчетов уже содержится в памяти ЦВМ, а результаты просто накапливаются в памяти. При такой обработке можно представить себе две совершенно независимые программы: одна для ввода массива из 1000 отсчетов, а вторая для вывода накопленных результатов в некоторое внешнее устройство. Очевидно, что время, затрачиваемое на выполнение этих программ, следует добавить ко времени счета, но никаких трудностей и усложнений в программе при этом не возникает. Однако при обработке в реальном времени взаимодействие между программами ввода — вывода и счета усложняется, и это часто приводит к заметному снижению быстродействия.

Для иллюстрации возникающих трудностей представим себе элементарную систему ввода — вывода, добавленную к основной схеме вычислительной машины, показанной на фиг. 11.1. Она содержит дополнительный регистр B, соединенный как с блоком памяти, так и с внешними устройствами. При выполнении команд ВВОД и ВЫВОД в регистр В начинает поступать информация (по команде ВВОД — от внешнего устройства). При этом вычисления прерываются до окончания приема данных в регистр В или выдачи их из регистра. После этого ЦВМ переходит к выполнению следующей команды, причем приведение входных и выходных данных к нужному формату осуществляется программным путем. Очевидно, что приведенная простейшая система ввода — вывода (как и вся элементарная вычислительная машина) малопригодна для практических целей, так как не позволяет автоматизировать прерывания или совместить выполнение программ счета и ввода — вывода. Однако с ее помощью можно оценить усложнение программы и дополнительные затраты времени при переходе к обработке в реальном времени.



Фиг. 11.4. Пример формата размещения данных в ЗУ для программы вычисления скользящей суммы в реальном времени ($k = 6$).

Покажем, как наличие простой системы ввода — вывода влияет на вычисление скользящей суммы. Пусть последовательность входных чисел циркулирует в памяти ЦВМ согласно фиг. 11.4. Вычисления проводятся следующим образом. Каждый новый отсчет, поступающий из регистра B, принимается за $x\,(n)$ и сначала запоминается в промежуточном регистре TEMP. Метка на схеме размещения данных в ЗУ (фиг. 11.4) соответствует отсчету $x\,(n - k)$ (для упрощения пояснений, относящихся к нашей простой системе ввода—вывода, k на фиг. 11.4 уменьшено до 6), причем ЗУ содержит ровно k регистров. Сначала вычисляется разность $x\,(n) - x\,(n - k)$, затем к ней добавляется значение $y\,(n - 1)$, находившееся в другом промежуточном регистре YN, а результат засылается в тот же регистр YN. В конце цикла отсчет $x\,(n - k)$ заменяется на $x\,(n)$, индекс увеличивается на единицу и выполняется команда ВЫВОД.

Ниже приведена программа вычисления скользящей суммы в реальном времени (снова принято $k = 26$):

	YIX	−25	Занести −25 в индексный регистр
	CLEAR	YN	
1.	INPUT		
2.	STORE B	TEMP	Занести содержимое регистра ввода—вывода [т. е. очередной отсчет $x\,(n)$] в промежуточный регистр
3.	SUB_x	XN+25	Вычесть $x\,(n-k)$ из $x\,(n)$
4.	ADD	YN	Добавить $y\,(n-1)$, что дает $y\,(n)$
5.	STORE	YN	Запомнить $y\,(n)$ в регистре YN, чтобы использовать в следующей операции
6.	LOAD	TEMP	
7.	STORE_x	XN+25	
8.	JNX	10.	
9.	YIX	−25	Восстановить исходное состояние индексного регистра.

10.	LOAD B	YN	Передать значение $y\,(n)$ в регистр ввода—вывода
11.	OUTPUT		
	JMP	1.	

Чтобы оценить быстродействие данной программы, положим, что при выполнении команды ВВОД машина ожидает появления синхросигнала от устройства ввода, затем выполняются команды 1 — 11, после чего работа ЦВМ прерывается до поступления синхросигнала от устройства вывода. Для увеличения быстродействия необходимо подобрать вполне определенную задержку между входным и выходным синхросигналами. Это означает, что при изменении программы задержку придется перестраивать. Если оба синхросигнала не имеют фазовой синхронизации, то максимальную скорость поступления данных придется уменьшить, чтобы учесть возможное удлинение остановок при выполнении команд ВВОД и ВЫВОД. Но даже при наличии синхронизации программа обработки в реальном времени выполняется примерно вдвое медленнее, чем программа обработки в произвольном масштабе времени.

11.6. Методы увеличения быстродействия вычислительной машины

Для увеличения быстродействия вычислительных машин используется множество вариантов небольшого количества основных принципов. Ряд методов связан с использованием блоков сверхоперативной памяти, обычно представляющих собой ЗУ небольшой емкости, но весьма быстродействующие, определенным образом вводимые в структуру ЦВМ. Идея здесь в том, что при составлении программы следует предусмотреть выполнение большинства медленных операций с применением сверхоперативного ЗУ (если это возможно). Другой ряд методов увеличения быстродействия ЦВМ связан с распараллеливанием арифметических операций. Эта идея имеет много разновидностей, поскольку вся конструкция ЦВМ зависит от степени параллелизма. Кроме того, распараллеливание управления, ввода — вывода и арифметических операций дает возможность уменьшить число машинных циклов, необходимых для выполнения данного алгоритма. К рассматриваемому типу распараллеливания относится перекрытие командных циклов с циклами поступления информации и с выполнением микропрограмм. Другой вид распараллеливания, при котором операции выполняются последовательно на одном потоке, называется *поточной обработкой*. Каждый новый отсчет начинает обрабатываться сразу же после окончания первой операции над предыдущим отсчетом, т. е. быстродействие системы значительно повышается.

Перечисленные идеи основаны на использовании в определенных блоках ЦВМ схемных элементов с соответствующими быстродействием и стоимостью. Согласование быстродействия элементов со структурой вычислительной машины является непременным условием правильного проектирования.

11.7. Сверхоперативные ЗУ

Существует несколько способов использования сверхоперативной памяти в составе ЦВМ:

1. В качестве логической части арифметического устройства (АУ).
2. В качестве устройства ввода—вывода.
3. В качестве основного ЗУ ЦВМ в сочетании с менее быстродействующим ЗУ большего объема, используемым для ввода—вывода.
4. Логическое объединение ее с оперативной памятью ЦВМ и выделение для нее некоторой части адресного поля.

В современных быстродействующих ЦВМ сверхоперативная память чаще используется в составе АУ. Типичными примерами таких ЦВМ являются CSP-30, содержащая 32 «сверхоперативных» регистра, а также META-4 и LX-1, имеющие по 16 регистров. Малые ЦВМ серии Nova имеют четыре, а PDP-11 — восемь регистров. Такие элементы памяти обычно называют общими регистрами, поскольку их можно использовать в различных целях: при выполнении арифметических операций, для ввода — вывода и индексации.

Второй из перечисленных способов использования сверхоперативных ЗУ применяется только в непрограммируемых внешних устройствах типа специализированных фурье-процессоров, рассмотренных в гл. 10. В таких системах основная ЦВМ может устанавливать специальное устройство управления в режим выполнения с большой скоростью некоторых определенных операций. Числа передаются из основной памяти в сверхоперативную, обрабатываются с помощью специализированного АУ, а результат возвращается в основную память для дальнейшей обработки или контроля.

Если объем сверхоперативной памяти достаточно велик, может оказаться целесообразным объединение основных устройств ЦВМ вокруг этой памяти. В данном случае память большего объема используется в основном как буфер аналогично магнитным барабанам, а вся работа выполняется с помощью сверхоперативной памяти. Хотя такая система и должна иметь быстродействующие устройства управления и АУ, однако структурно она ближе к универсальной высокопроизводительной вычислительной машине для обработки сигналов.

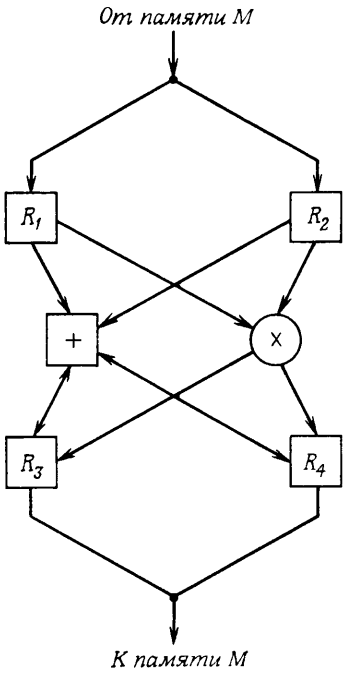
Можно, наконец, объединить быстрое и медленное ЗУ в единую логическую структуру. Для обеспечения ее эффективности необходимо, чтобы устройство управления различало, какая из областей памяти участвует в вычислениях, и соответственно изменяло частоту синхронизации. Кроме того, по-прежнему и арифметическое, и управляющее устройства должны иметь достаточно высокое быстродействие.

Необходимо отметить, что каждому из приведенных выше методов свойствен компромисс между функциональными возможностями и затратами на его реализацию. Различные методы можно также применять совместно. В данной главе будет показано, как эти возможности используются в различных ЦВМ.

11.8. Распараллеливание арифметических операций

Не вызывает сомнения, что, удвоив объем вычислительного оборудования, можно увеличить быстродействие системы, однако на практике реализовать потенциальные возможности такого распараллеливания совсем непросто. Давно известным, но весьма эффективным методом введения (при необходимости) распараллеливания арифметики является байтовая организация регистров ЦВМ. Так, ЦВМ Линкольновской лаборатории TX-2 оперирует с 36-разрядными числами, составленными из четырех байтов длиной по девять разрядов каждый. Арифметическое устройство также имеет байтовую структуру, так что с помощью одной команды можно перемножить, например, пару 36-разрядных чисел, две пары 18-разрядных или 4 пары 9-разрядных. Используя программные хитрости, в такой структуре можно существенно повысить скорость обработки, если только допустимо применение малоразрядных чисел.

Увеличения степени параллелизма можно достичь, используя несколько арифметических устройств, каждое из которых оперирует с полноразрядными числами. Примером такой системы является быстродействующий цифровой процессор (FDP — Fast Digital Processor) Линкольновской лаборатории, содержащий четыре 18-разрядных АУ и два 18-разрядных ЗУ. При таком распараллеливании программирование, позволяющее использовать возможности машины, значительно усложняется. Основная трудность состоит в необходимости начальной установки блоков и в сложности вывода результатов. Перед началом совместной работы все АУ должны быть установлены в исходное состояние, а это требует времени. Результаты параллельных вычислений нужно записать в соответствующие места, на что также уходит время. Дополнительные трудности возникают, если различные АУ должны быть взаимосвязаны. Число возможных связей между n арифметическими устройствами растет как $n!$, поэтому число управляющих



Фиг. 11.5. Арифметическое устройство для обработки комплексных чисел.

цепей, если не ограничивать эти связи, может выйти из-под контроля. Вынужденные ограничения в числе взаимосвязей превращают обмен данными в продолжительную и сложную процедуру.

Приведем несколько примеров систем с параллельной арифметикой, предназначенных для эффективного выполнения алгоритмов БПФ. Поскольку эти алгоритмы оперируют с комплексными числами, рассмотрим, каким образом «комплексное» АУ (фиг. 11.5) позволяет увеличить скорость вычислений. Предположим, что регистры R_1, R_2, R_3 и R_4 имеют двойную длину (т. е. предназначены для комплексных чисел), а сумматор и умножитель оперируют с комплексными числами. Ниже приведена программа выполнения базовой операции БПФ $A' = A + CW^k$, $C' = A - CW^k$, причем предполагается, что перед началом вычислений в регистре R_2 содержится число W^k:

Программа	*Интерпретация*
1. $M \to R_1$	$C \to R_1$
2. $R_1 \times R_2 \to R_4$	$CW^k \to R_4$
3. $M \to R_1$	$A \to R_1$
4. $R_1 + R_4 \to R_3$	$A' = A + CW^k \to R_3$
5. $R_1 - R_4 \to R_4$	$C' = A - CW^k \to R_4$
6. $R_3 \to M$	Запомнить A'
7. $R_4 \to M$	Запомнить C'

В программу включены операции обмена с памятью и арифметические операции, используемые в варианте алгоритма БПФ с основанием 2 и прореживанием по времени. Операции над индексами здесь опущены; для их выполнения могут понадобиться, скажем, две дополнительные команды или же их можно выполнять параллельно с вычислениями и обращениями к памяти.

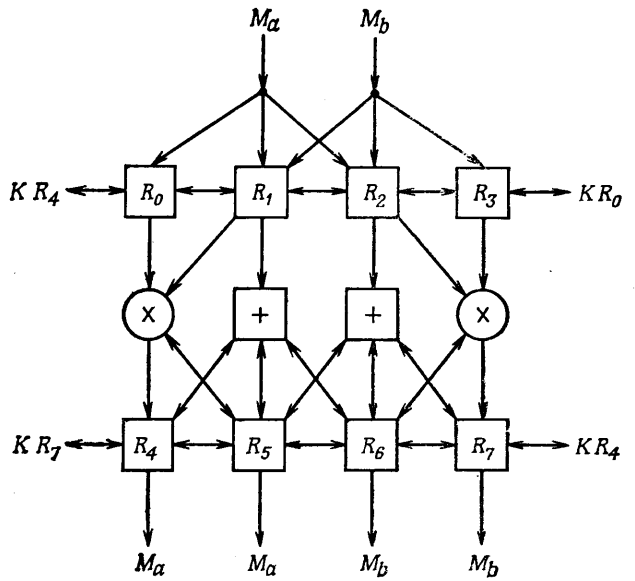
За счет дополнительного усложнения устройства управления количество команд в цикле можно уменьшить, совмещая обращение к памяти с арифметическими операциями:

Программа		*Интерпретация*
1. $M \to R_1$		$C \to R_1$
2. $R_1 \times R_2 \to R_4$	$M \to R_1$	$CW^k \to R_4$ $A \to R_1$
3. $R_1 + R_4 \to R_3$		$A' = A + CW^k \to R_3$
4. $R_1 - R_4 \to R_4$	$R_2 \to M$	$C' = A - CW^k \to R_4$, запомнить A'
5. $R_4 \to M$		Запомнить C'

Отметим, что для такого сокращения не требуются логические усложнения структур АУ и ЗУ. Так, при выполнении команды 2 не возникает аппаратурных трудностей, связанных с использованием регистра R_1 как для передачи чисел в умножитель, так и для приема их из памяти. Последнее возможно, поскольку между выходными и входными сигналами триггеров регистра всегда существует задержка.

Комплексное АУ (фиг. 11.5) содержит четыре умножителя (действительных чисел) и четыре сумматора (два для выполнения комплексного умножения и два для комплексного сложения). Вполне уместно спросить: целесообразно ли такое усложнение аппаратуры для ускорения выполнения базовой операции БПФ? Для сравнения на фиг. 11.6 показано АУ, содержащее два сумматора, два умножителя и те же восемь регистров.

Упражнение. Предполагая, что в схеме фиг. 11.6 арифметические операции и обращения к памяти могут выполняться па-



Фиг. 11.6. Арифметическое устройство с восемью регистрами.

раллельно, определите число команд, необходимых для вычисления одной базовой операции БПФ с основанием 2 и прореживанием во времени. Запишите эту программу.

11.9. Параллельная работа ЗУ, АУ, устройств управления и вызова команд

Каждая исполняемая команда управляет совокупностью вентилей и триггеров, в результате чего в ЦВМ выполняется определенная последовательность операций. Многие из этих операций могут выполняться одновременно, причем логический ход программы не будет нарушаться. За выигрыш в быстродействии, получающийся при этом, приходится расплачиваться усложнением устройства управления. Так, операции перехода, занесения в память и сложения можно было бы совместить в одной команде. Логических прерываний в работе при этом происходить не будет, но длина кода команды становится весьма большой. В качестве примера предположим, что адреса чисел и команд содержат по 12 разрядов, а коды операций — по шесть разрядов. В итоге длина кода команды, содержащей три кода операций и два адреса, должна быть равна 42 двоичным разрядам.

Дополнительное распараллеливание (которое можно также рассматривать как поточную обработку) получается при параллельной работе ЗУ команд и ЗУ чисел, так что выполнение очередной команды может быть начато до завершения предыдущей. При этом возникает опасность, что две команды могут потребовать от одного и того же устройства выполнения различных действий. Другая опасность состоит в том, что некоторая условная команда (например, условный переход) может быть выполнена неправильно, так как к началу ее выполнения еще не была закончена предыдущая команда, которая может изменить условие перехода.

Существует несколько методов, позволяющих избежать таких ошибок. Один из них — введение блокирующей цепи, обнаруживающей ситуации, ведущие к ошибкам, и соответственно задерживающей вызов и исполнение последующих команд. Второй метод — аккуратная проработка и синхронизация последовательности операций для каждой команды и введение определенных правил программирования. Этот метод будет продемонстрирован в разд. 11.10 при рассмотрении FDP.

11.10. Быстродействующий цифровой процессор (FDP) Линкольновской лаборатории

Причиной создания FDP послужили исследования в области сжатия речевых сигналов, проводившиеся в Линкольновской лаборатории. К этому времени стало ясно, что перед созданием реальной системы с вокодером (см. гл. 12) весьма полезно промоделировать работу систему на ЦВМ. Основным недостатком машинного моделирования была невозможность проведения обработки в реальном времени. Мнение человека, воспринимающего информацию при прослушивании речи или при наблюдении за индикаторами радио- или гидролокатора, во многом зависит от условий получения этой информации. Если система не была испытана в реальных условиях, нельзя получить оценку качества ее работы, которую оператор мог бы дать, опираясь на свою интуицию.

Предполагалось, что при использовании идеи сверхоперативной памяти и быстродействующих микросхем эмиттерно-связанной логики можно достичь примерно десятикратного увеличения быстродействия. Однако для моделирования системы обработки речи в реальном времени требовалось повышение быстродействия еще на порядок (цифры относятся к использовавшейся в то время мини-ЭВМ типа Univac 1219). Это означало, что необходимо было использовать все возможности увеличения быстродействия:

1) распараллеливание арифметики;
2) распараллеливание обращения к памяти;
3) перекрытие операций с числами и подготовка к исполнению команд;
4) использование многоцелевых команд;
5) параллельное управление адресацией, памятью и вычислениями;
6) умножение в отдельном функциональном блоке.

Полная структурная схема FDP изображена на фиг. 11.7. Следует отметить некоторые ее особенности:

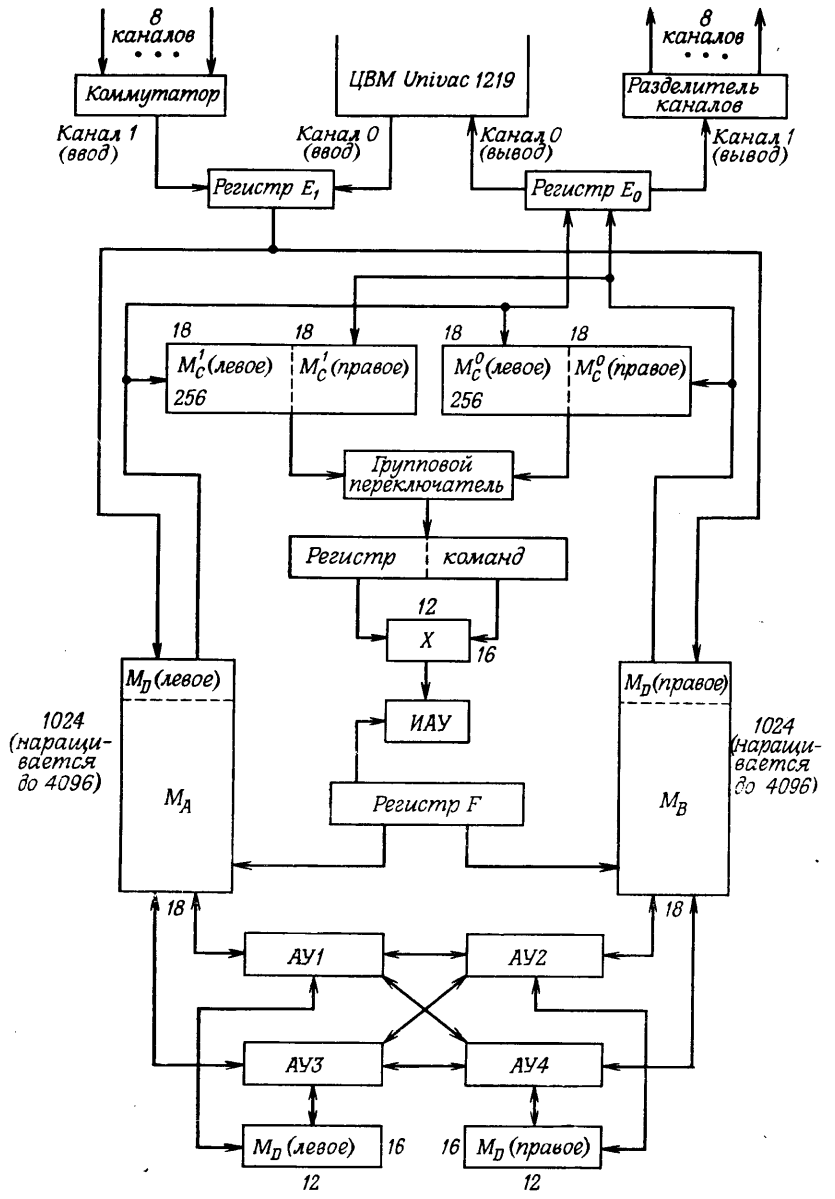
1. Наличие двух ЗУ с независимой адресацией, выполненных на интегральных схемах. Цикл памяти составляет 150 нс, а объем — по 4096 слов (регистров). Роль основной памяти играет оперативная память мини-ЦВМ Univac 1219.

2. Наличие четырех АУ с независимым управлением, каждое из которых содержит сумматор, умножитель в виде независимого функционального блока и три программируемых регистра.

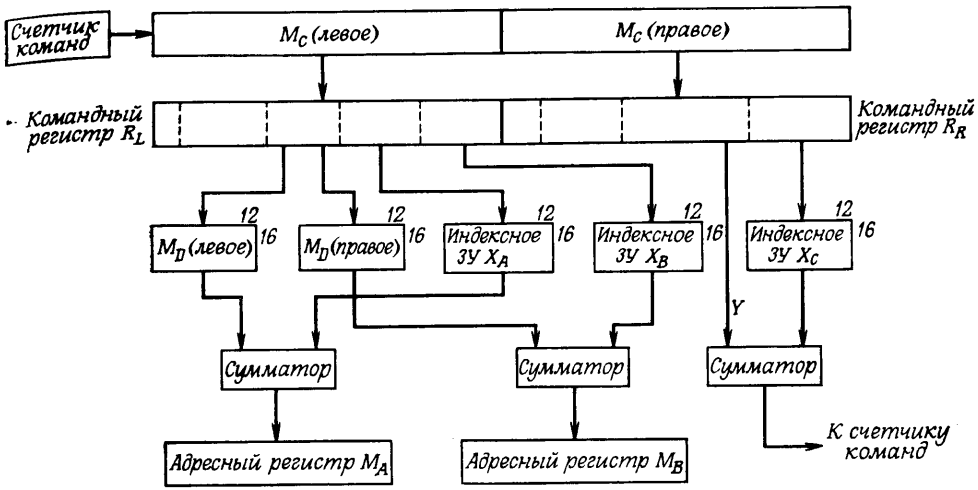
3. ЗУ для хранения программ (объемом 512 слов) способно выдавать 36-разрядные команды через интервалы 150 нс.

4. Пятое арифметическое устройство (с меньшими возможностями) ИАУ (индексное АУ) предназначено для обслуживания индексных регистров. Связь между ним и четырьмя большими АУ обеспечивается через ЗУ данных M_A и M_B, а также через регистр F.

Адресация осуществляется согласно фиг. 11.8. Команды с обращением к ЗУ M_A и M_B должны исходить из ЗУ M_C (левого), а команды с обращением к M_C — из ЗУ M_C (правого). Четыре разряда левого регистра команд IR_A используются для адресации в ЗУ M_D, из которого извлекаются базовые адреса для ЗУ M_A и M_B.



Фиг. 11.7. Общая блок-схема процессора FDP (длина слов показана вдоль горизонтального размера ЗУ, а число слов — вдоль вертикального размера ЗУ).



Фиг. 11.8. Адресация в процессоре FDP.

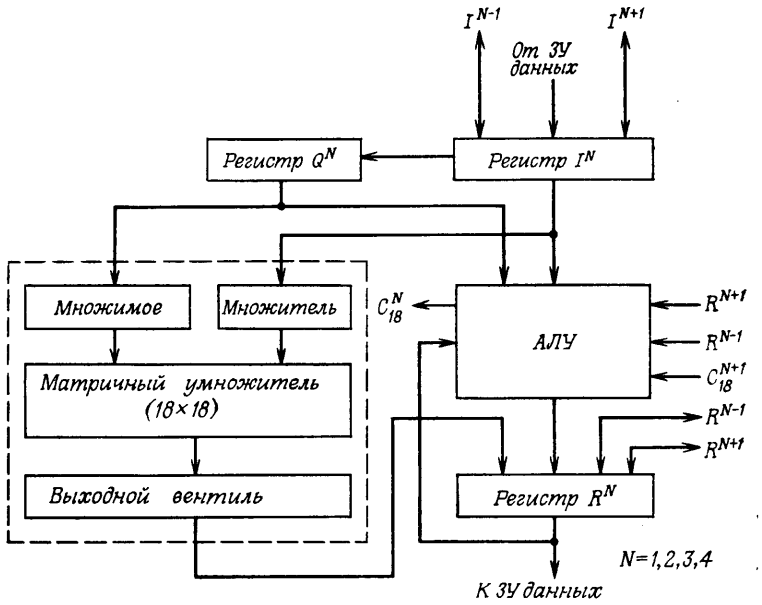
Эти базовые адреса уточняются с помощью индексных ЗУ X_A и X_B соответственно.

При обращении к ЗУ M_C адрес образуется с помощью восьмиразрядного адреса Y, хранящегося в индексном регистре IR_B. Для некоторых команд перехода к адресу Y добавляется содержимое регистра X_C. Хотя чтение из всех трех индексных ЗУ может выполняться независимо, при записи в ЗУ адреса обращения к ним должны быть одинаковыми. Таким образом, с точки зрения программиста в FDP имеется одно индексное запоминающее устройство X, к которому с целью модификации адресов M_B и M_C можно обращаться с помощью трех адресов, содержащихся в двойной команде.

11.11. Структурные схемы арифметических устройств

Структура и связи одного из четырех одинаковых арифметических устройств FDP представлены на фиг. 11.9. Регистры I, Q и R являются программно-управляемыми. Арифметическое устройство АУ1 соединено с АУ2 и АУ4; АУ2 — с АУ3 и АУ1; АУ3 — с АУ4 и АУ2; АУ4 — с АУ1 и АУ3. Таким образом, все арифметические устройства соединены в кольцо с $n = 4$ и $R^{n+1} = R^1$. Регистр R^n можно рассматривать как накопитель, поскольку в него поступают результаты как сложения, так и умножения.

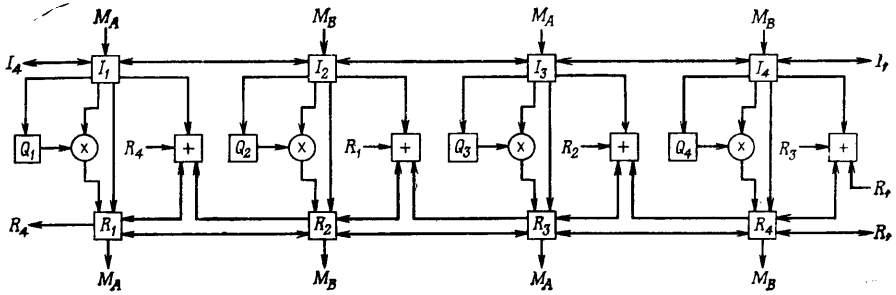
Отметим существенные особенности АУ:



Фиг. 11.9. Линии связи в процессоре FDP.

1. Каждое АУ можно программировать независимо от других АУ и параллельно им. Так, с помощью одной 18-разрядной кокоманды можно выполнить в АУ1 умножение, в АУ2 передачу из регистра I^2 в регистр R^2 и т. д.

2. Команда умножения является в сущности командой *передачи*, согласно которой содержимое регистров I^n и Q^n передается в соответствующие им регистры умножителя, а программируемые



Фиг. 11.10. Программно-управляемые линии связи всех четырех АУ в процессоре FDP.

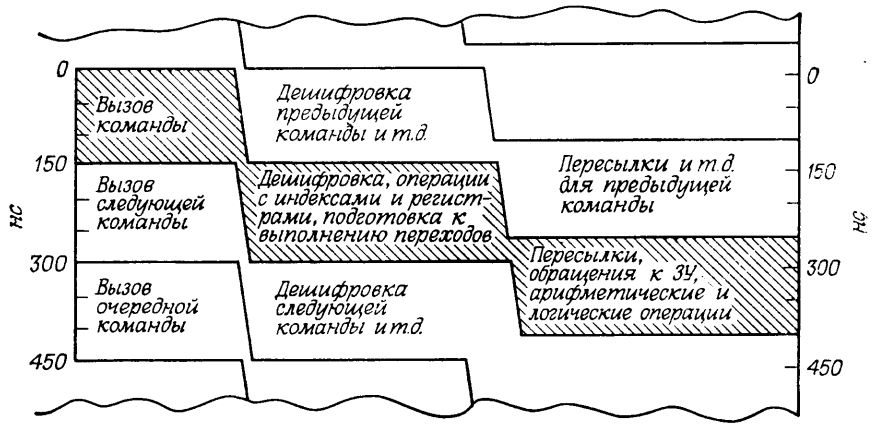
регистры I^n, Q^n и R^n освобождаются, вследствие чего во время умножения n-е АУ может выполнять другие команды. Время умножения в существующем FDP (включая начальную команду MUL) равно длительности исполнения четырех команд. Таким образом, каждый умножитель является как бы специальным дополнительным устройством к соответствующему АУ и не вызывает прерывания его работы до окончания умножения. На фиг. 11.10 показано управление линиями связи всех четырех АУ.

11.12. Синхронизация

На фиг. 11.11 представлена диаграмма синхронизации команд. Время выполнения каждой команды равно 400 нс, но за счет одновременного выполнения нескольких команд эффективное время выполнения команды равно 150 нс. В любой момент времени выполняются те или иные действия, относящиеся к трем различным последовательным командам. Команда выполняется в три этапа:

1. Вызов команды (из ЗУ M_C).
2. Декодирование команды, операции с индексными регистрами, принятие решений о переходах.
3. Пересылки, обращения к памяти, арифметические и логические операции.

На каждом интервале длительностью 150 нс выполняются операции всех трех этапов, но относящиеся, конечно, к трем разным последовательным командам. Поскольку команды вызываются через каждые 150 нс, *эффективное* быстродействие определяется именно этой величиной, хотя фактическое время выполнения каждой команды равно 400 нс (для выполнения некоторых специальных



Фиг. 11.11. Временная диаграмма выполнения команд в процессоре FDP.

команд требуется больше 400 нс, но основной интервал синхронизации не меняется).

Вследствие рассмотренного временно́го перекрытия операции типа перехода, синхронизируемые таким же образом, могут изменить последовательность выполнения команд только после вызова команды, следующей непосредственно за командой перехода. Для повышения эффективности эта последующая команда не прерывается, а выполняется до конца, и поэтому переход к новым адресам в M_C начинается через одну команду после команды перехода. Таким образом, команда перехода, завершающая некоторый цикл, должна быть в этом цикле не последней, а предпоследней.

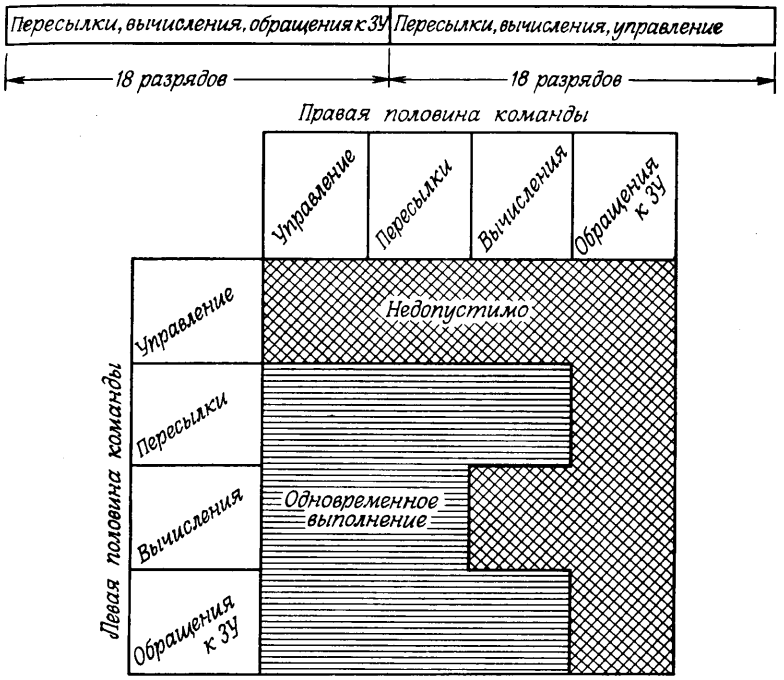
Команды типа пропуска (SKIP) отменяют исполнение следующей за ними команды, так что время, затрачиваемое на операции с этой последующей командой, не зависит от результата выполнения команды пропуска. Специальная команда (находящаяся в левой половине кода двойной команды) пропуска при переходе SOJ (skip on jump), не связанная с обращением к ЗУ, означает, что если правая половина команды соответствует переходу, то при выполнении перехода последующая команда отменяется. Таким образом, команда SOJ в сочетании с командой перехода выполняется как обычная команда перехода, но при этом затрачивается 300 нс, если переход происходит, и 150 нс, если перехода не происходит.

11.13. Обзор методов увеличения быстродействия, использованных в FDP

Перечислим методы, использованные в FDP для ускорения обработки:

1. Параллельная работа ЗУ данных M_A и M_B позволяет вдвое сократить число циклов обращения к памяти. Так, во внутреннем цикле БПФ в арифметическое устройство, выполняющее базовую операцию БПФ, необходимо ввести два комплексных числа и затем вывести в ЗУ два комплексных результата. В FDP для этого используются четыре цикла обращения к памяти, тогда как без распараллеливания ЗУ потребовалось бы восемь циклов.

2. Поточное выполнение команд позволяет вызывать их через каждые 150 нс. Все команды (кроме умножения) полностью выполняются за 400 нс, т. е. поточная обработка приводит к $^8/_3$-кратному увеличению быстродействия. При проектировании FDP было принято, что поточная обработка используется для всех команд. Это облегчает определение времени выполнения любых программ. С помощью дополнительных аппаратурных связей можно было бы проводить обработку поточным способом только для определенных последовательностей команд и видов данных. Недостаток такого подхода заключается в усложнении расчета



Фиг. 11.12. Допустимые сочетания команд.

времени выполнения программы, что затрудняет программирование задач обработки в реальном времени.

3. Применение многоцелевых команд сокращает необходимое число тактов, так как некоторые команды, например команды обращения к памяти и условного перехода, можно вызывать и выполнять одновременно. Понятно, что выполнение команды условного перехода определяется результатами предыдущих команд, но не зависит от команды, выполняемой одновременно с ней. На фиг. 11.12 приведены допустимые сочетания операций, задаваемых левой и правой половинами 36-разрядного слова кода команды.

4. Четырехкратное распараллеливание арифметики также уменьшает число тактов, хотя в этом случае в отличие от других перечисленных методов ускорения процессора дать количественную оценку увеличения быстродействия довольно трудно. Такую оценку можно получить непосредственным расчетом, составив программы для типичного алгоритма и рассчитав время их выполнения в предположении, что используются одно, два или четыре АУ.

В качестве простейшего примера рассмотрим задачу суммирования N чисел. Допустим сначала, что имеются всего одно ЗУ и одно АУ, причем регистр R очищен. Программа в этом случае имеет вид

```
        M_A → I   ;  NO-OP [1])
BACK →  M_A → I   ;  JNX  BACK
        NO-OP     ;  I+R → R
                     HALT
```

Символ JNX означает условный переход (и добавление к индексу единицы), если в индексном регистре X содержится отрицательное число; в противном случае переход не выполняется. Главная особенность этой несложной программы состоит в том, что третья строка выполняется раньше команды условного перехода, записанной во второй строке. Таким образом, внутренний цикл содержит две команды и выполняется за 300 нс.

Предположим теперь, что имеются два ЗУ (M_A и M_B) и два АУ (АУ1 и АУ2), причем регистры R_1 и R_2 очищены. Пусть, кроме того, суммируемый массив разделен на две половины, содержащие по $N/2$ чисел и хранящиеся в ЗУ M_A и M_B. Программа имеет вид

```
          M_A → I_1, M_B → I_2  ;  NO-OP
BACK →    M_A → I_1, M_B → I_2  ;  JNX  BACK
          NO-OP                 ;  I_1+R_1 → R_1, I_2+R_2 → R_2
                                   HALT
```

Внутренний цикл снова состоит из двух команд, но, поскольку АУ1 и АУ2 работают параллельно, число его повторений уменьшается вдвое; после окончания цикла рассчитывается сумма $R_1 + R_2$. Таким образом, удвоение числа арифметических и запоминающих устройств почти вдвое увеличивает быстродействие.

При наличии четырех АУ программа имеет вид

```
          M_A → I_1, M_B → I_2  ;  NO-OP
          M_A → I_3, M_B → I_4  ;  NO-OP
          M_A → I_1, M_B → I_2  ;  R_n+I_n → R_n для всех n
BACK →    M_A → I_3, M_B → I_4  ;  JNX BACK
          M_A → I_1, M_B → I_2  ;  R_n+I_n → R_n
                                   HALT
```

[1]) NO-OP — неисполняемая команда.— *Прим. ред.*

Примечательно, что внутренний цикл по-прежнему состоит из двух команд, т. е. программа выполняется вдвое быстрее предыдущей и в четыре раза быстрее программы для одного АУ. Эти простые примеры подтверждают, что способ задания команд, принятый в FDP, позволяет эффективно использовать возможности параллельной арифметики FDP.

Во всех приведенных программах опущены команды адресации памяти и установки нужных значений индексов. Однако в данном элементарном случае никаких дополнительных команд не требуется.

11.14. Выполнение быстрого преобразования Фурье с помощью FDP

При проектировании FDP одной из главных задач было обеспечение наибольшей универсальности машины и одновременно максимальной ее приспособленности к выполнению алгоритма БПФ. Полезно рассмотреть, какие операции составляют внутренний цикл БПФ, а также выяснить, насколько компактной получается программа на языке программирования, принятом в FDP. Предположим сначала, что коэффициенты хранятся в регистрах Q. Тогда для ввода чисел в АУ и вывода результатов в ЗУ необходимы два цикла чтения и два цикла записи, т. е. всего четыре команды обращения к памяти. Для выполнения всех четырех умножений достаточно одной команды, но длится умножение три (двойных) такта. Необходимо также выполнить шесть сложений — два при комплексном умножении и четыре при сложении двух пар комплексных чисел. Для этого нужны еще две команды. Кроме того, пришлось ввести три команды для индексации, одну команду условного перехода и одну для проверки переполнения. Объединим эти данные в таблицу.

Тип команды	Количество команд
Обращение к памяти	4
Умножение	6
Сложение	2
Индексация	3
Условный переход	1
Проверка на переполнение	1
Всего	17

Оказывается, что внутренний цикл на языке FDP можно записать в восьми строках (без проверки на переполнение достаточно пяти строк), т. е. в виде 16 команд. Это представляется невозможным, так как в таблице перечислено 17 команд, или 8,5 строк. Но если вспомнить, что умножитель имеет свои регистры и умножение происходит в отдельном блоке, так что арифметические устройства в это время могут выполнять другие действия, то становится понятным, что одновременно с выполнением одной базовой операции БПФ возможно исполнение некоторых команд последующей базовой операции.

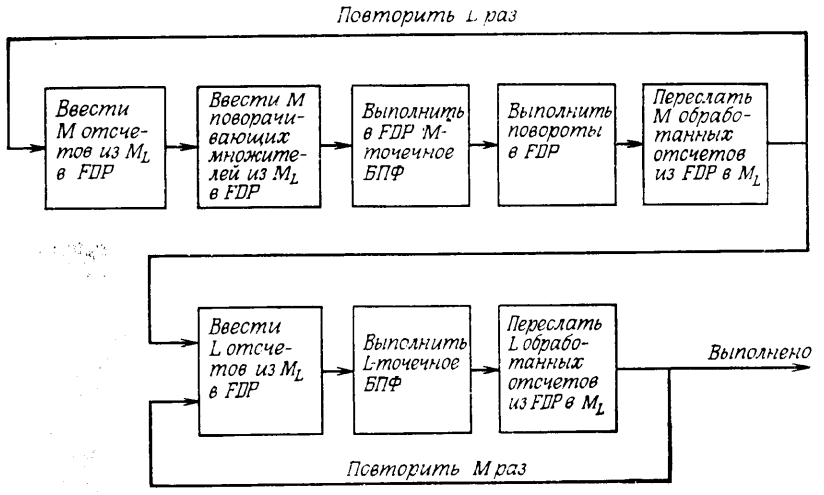
Насколько в структурах типа FDP можно ускорить выполнение базовой операции? Мы видим, что при этом необходимы четыре обращения к памяти (два для считывания и два для записи). Следовательно, FDP всего вдвое уступает в быстродействии специализированным устройствам, построенным на тех же элементах и имеющим аналогичную структуру. Ниже (при рассмотрении LSP2) будет показано, что можно создать как специализированные, так и универсальные вычислительные устройства с бо́льшим быстродействием.

Для обработки сигналов может быть использована также программа рекурсивной цифровой фильтрации. FDP, являющийся программируемым вычислительным устройством, позволяет выполнить обработку по любому подобному алгоритму. В качестве оценки среднего числа команд, необходимых для моделирования цифрового резонатора с двумя полюсами, можно принять восемь команд (или 1,2 мкс). Интересно отметить, что расчеты, связанные с моделированием фильтра связанной формы, занимают не больше времени, чем для фильтров прямой или канонической формы, хотя для первой требуется двое больше умножений. Объясняется это тем, что все четыре умножителя могут работать одновременно.

11.15. Подпрограммы для действий с плавающей запятой

FDP оперирует 18-разрядными числами с фиксированной запятой, но позволяет также выполнять действия над числами, имеющими плавающую запятую. В последнем случае сложение занимает около 6,5 мкс, изменение знака — 1,5 мкс, умножение — около 2,0 мкс, деление — около 10 мкс. Эти величины приблизительные, так как продолжительность операции зависит от значений обрабатываемых чисел.

Под обрабатываемые массивы в FDP отводятся два блока памяти M_A и M_B объемом по 4096 слов каждый. Отсюда следует, что БПФ массива, содержащего более 2048 точек, нельзя выполнить без обращения к большей памяти. Роль этой дополнительной памяти большего объема играет ЗУ M_L, подключенное к каналу ввода—вывода FDP. Рассмотрим, как выполняется БПФ с использованием



Фиг. 11.13. Вычисление БПФ большого массива с использованием магнитного ЗУ большого объема в составе FDP.

ЗУ M_L. В гл. 6 было показано, что если представить данные в виде двумерной матрицы размером $L \times M$, причем $N = LM$, то БПФ всего массива можно получить, найдя БПФ отдельных строк, перемножив все коэффициенты на поворачивающие множители и выполнив затем БПФ столбцов. Поскольку поворачивающих множителей N, их следует хранить в M_L вместе с обрабатываемым массивом и при считывании очередной строки данных в ЗУ FDP вместе с ней должна передаваться и соответствующая строка поворачивающих множителей. После поворота эта строка пересылается обратно в M_L. Таким образом, для обработки (включая повороты всех строк матрицы) требуется $3N$ циклов обращения к памяти M_L. Как видно из фиг. 11.13, еще $2N$ циклов потребуется для вычисления БПФ столбцов и пересылки результатов в память M_L. Полагая, что поворот занимает столько же времени, что и выполнение базовой операции, и обозначая это время через τ_b, а время, затрачиваемое на обмен информацией между FDP и M_L, — через T_m, получим следующее выражение для времени выполнения БПФ всего массива:

$$T = \frac{N\tau_b}{2}(\log_2 N + 2) + 5NT_m. \qquad (11.3)$$

Если, например, $N = 65\,536 = 2^{16}$ и $\tau_b = 1{,}2$ мкс, то $T = 0{,}87$ с, причем бо́льшую часть этого времени занимает выполнение базовых операций, необходимых в любом случае. Время, затрачиваемое только на выполнение $(N/2)\log_2 N$ базовых опера-

ций (если весь массив данных помещается в ЗУ FDP), составляет $T = 0{,}629$ с. Отсюда можно было бы сделать вывод, что ограничение объема быстродействующей памяти FDP не слишком замедляет вычисление БПФ больших массивов, когда массивы хранятся в большом ЗУ с произвольным доступом, подключенном к FDP в качестве устройства ввода—вывода. С таким заключением не следует, однако, спешить. Заметим, что если время T_m обращения к внешней памяти M_L слишком велико, так что $5NT_m$ превышает первое слагаемое суммы (11.3), то T_m становится ограничивающим фактором. В FDP T_m может составлять, например, 1,6 или 3,2 мкс (в зависимости от режима работы). При $T_m = 1{,}6$ мкс и $N = 65\,536$ время обмена $5NT_m = 0{,}52$ с, т. е. меньше времени счета и не играет существенной роли. При $T_m = 3{,}2$ мкс, однако, $5NT_m = 1{,}04$ с, т. е. именно оно ограничивает скорость обработки.

Упражнение. Пусть объем памяти FDP равен 1024 числам и необходимо выполнить 1024-точечное БПФ, используя M_L. Считая, что $\tau_b = 1{,}2$ мкс, $\tau_m = 0{,}5$ мкс и $T_m = 3{,}2$ мкс, определите время обработки. Сравните результат со временем вычисления БПФ того же массива в FDP при большем объеме быстродействующей памяти.

Упражнение. Несколько более сложным вариантом предыдущей задачи является выполнение БПФ большого объема в реальном времени. Попытайтесь перечислить основные действия (и определить длительность их выполнения), необходимые для такой обработки. Составьте блок-схему программы. В результате должна получиться система с одним входом и одним выходом, причем отсчеты сигнала последовательно поступают на вход, а отсчеты спектра последовательно появляются на выходе в реальном времени. Оцените максимальную скорость поступления данных, при которой FDP еще справляется с обработкой.

11.16. Обзор особенностей FDP, связанных с распараллеливанием

В FDP действия над командами, арифметические операции и обращения к памяти выполняются параллельно. В связи с этим пришлось изменить некоторые стандартные команды ЦВМ, а также ввести несколько новых команд. Выше уже упоминались затруднения при выполнении команд перехода, вызванные поточным исполнением команд. В качестве выхода из положения было решено переходить к выполнению команды, следующей за командой перехода, не дожидаясь окончания проверки условия перехода. Такая схема вполне допустима при наличии в тексте программы одиночных (изолированных) команд перехода. Если же встречаются цепочки переходов, то зависимость получаемых результатов от входных

(обрабатываемых) данных настолько сложна, что программисту будет трудно разобраться в работе машины. Для решения проблемы было предложено ввести команду «пропуск при переходе» SOJ (skip on jump), записываемую в левых 18 разрядах, если в правых 18 разрядах записана команда перехода. При этом, хотя проверка условия выполнения перехода еще не закончена, последующая команда вызывается, но пока не исполняется. Следовательно, сохраняется возможность отмены этой команды, если в результате проверки условия перехода выяснится, что переход необходим. Таким образом, за счет команды SOJ переход в FDP становится аналогичным переходу в обычной ЦВМ, но при этом приходится терять 150 нс на вызов отменяемой команды.

В большинстве ЦВМ переходы соответствуют различным арифметическим условиям: в накопителе находится нулевое или отрицательное число, произошло переполнение и т. д. Поскольку в FDP используются четыре арифметических устройства, то ситуация усложняется. Эта трудность преодолевается соответствующим выбором формата команды:

6	8				
Переход по арифметическому условию	Y	E_1	E_2	E_3	E_4

Четыре разряда справа определяют, какое АУ является управляющим. Если какое-либо из E_i равно 1, то переход произойдет, когда в соответствующем арифметическом устройстве $АУ_i$ будет выполнено условие перехода. Если несколько E_i равно 1, то переход происходит согласно логической функции ИЛИ. Так, если $E_1 = E_2 = E_3 = E_4 = 1$, то переход по команде JPR (jump if R is positive — переход при положительном R) произойдет, если хотя бы в одном из четырех регистров R окажется положительное число.

Другим методом обработки условий перехода в FDP является использование команды блокировки NUL (nullify). При параллельной работе четырех АУ может потребоваться, чтобы каждое АУ работало по своей программе, зависящей от получаемых результатов (данных). Команда NUL позволяет заблокировать любую совокупность АУ, в результате чего состояния этих АУ не будут изменяться, даже если на них будут поступать соответствующие команды. Код команды NUL имеет следующий формат:

6	3	3	3	2
NUL	АУ1	АУ2	АУ3	АУ4

Для каждого АУ возможны восемь условий блокировки:

000	не блокировать
001	блокировать, если R = 0
010	блокировать, если R ≠ 0

011 блокировать, если $R > 0$
100 блокировать, если $R \geqslant 0$
101 блокировать, если $R < 0$
110 блокировать, если $R \leqslant 0$
111 блокировать

В дополнение к команде NUL имеется команда ACT (activity) снятия блокировки с АУ, по условию или безусловно отменяющая блокировку АУ. Обе команды — NUL и ACT — исполняются с задержкой на время выполнения одной команды, что позволяет выполнить в АУ две команды после проверки условия выполнения команды NUL и одну после проверки условия выполнения команды ACT.

Еще одна трудность связана с разделением памяти на ЗУ программы и ЗУ чисел, а также с наличием синхронного поточного обращения к ЗУ M_C. Такое обращение может привести к сбоям, если разрешается как считывание, так и *запись* в M_C. Если же запись не разрешается, то программы нельзя изменять, а это весьма неудобно. Например, программы БПФ и обратного БПФ отличаются лишь пятью или шестью командами. Кроме того, программа может быть большой и не помещаться в 512 ячейках, имеющихся в M_C. Поэтому, чтобы скорость обработки существенно не изменялась, излишек текста программы следует хранить в ЗУ M_A и M_B, предусмотрев возможность быстрого считывания из этих ЗУ в M_C. В FDP это обеспечивается командой пересылки массива, при выполнении которой управление передается специальному устройству, единственной задачей которого является пересылка команд из ЗУ M_A и M_B в ЗУ M_C. Число пересылаемых команд и их размещение в ЗУ задаются кодом команды пересылки массивов.

При описании FDP уже отмечалось, что процессор оперирует с 18-разрядными словами. Следует, однако, помнить, что полная длина слова команды равна 36 разрядам, а так как к ЗУ M_A и M_B можно обращаться для записи или считывания одновременно, то в распоряжении программиста имеются и 36-разрядные числа. Более того, наличие четырех АУ позволяет формировать и обрабатывать 72-разрядные числа. Предусматривая все эти варианты, проектировщики постарались обеспечить в FDP широкие возможности для программного изменения точности вычислений. Для этого используется команда объединения LNK (link). С ее помощью можно разряд переноса передать из $АУ_{n+1}$ в $АУ_n$, а с применением команд сдвига — перемещать числа из $АУ_{n+1}$ в $АУ_n$ и обратно. Например, команда LNK используется для сложения 36-разрядных чисел. Первыми складываются младшие 18 разрядов, а затем — старшие. Если в команде LNK все четыре определяющих разряда — единицы, то все 72 разряда в четырех регистрах R можно сдвинуть (по кольцу) вправо или влево на один разряд, используя только одну 18-разрядную команду.

11.17. Процессор Линкольновской лаборатории LSP2 (Lincoln Signal Processor 2) для обработки сигналов

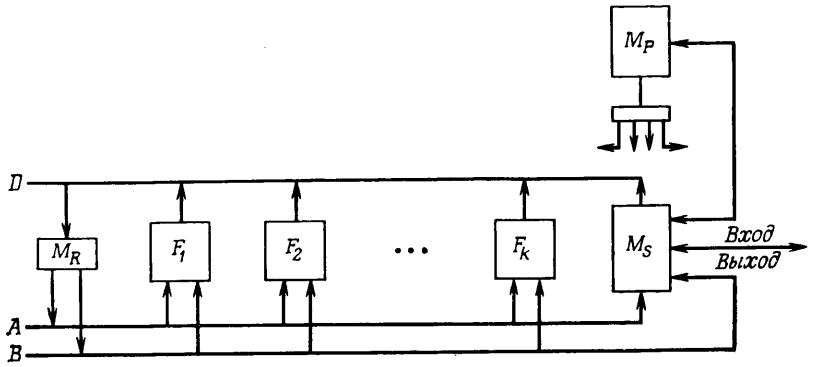
Разработка FDP в значительной мере была экспериментом, позволившим накопить опыт применения параллелизма при обработке сигналов. В целом эту разработку можно считать успешной и полезной, так как она доказала, что подобное сочетание гибкости и быстродействия в среднем более выгодно, чем высокое быстродействие без гибкости или гибкость без быстродействия. Кроме того, приобретенный опыт позволяет разрабатывать и более эффективные системы обработки сигналов. Чтобы показать это, рассмотрим сначала структуру FDP. Во-первых, следует отметить, что в FDP, как и в большинстве универсальных ЦВМ, много времени расходуется на обращение к памяти, индексацию и переходы. Во-вторых, для эффективного использования четырех параллельных АУ приходится с большими затратами времени составлять весьма сложные программы. (Накопление библиотеки стандартных программ впоследствии ускорит процесс программирования.) В-третьих, FDP построен на базе ЭСЛ-микросхем с невысокой степенью интеграции, быстродействие которых примерно вдвое ниже, чем у современных улучшенных ЭСЛ-микросхем. Предварительный анализ показал, что можно построить процессор большей мощности, более дешевый и компактный и проще программируемый, чем FDP. В-четвертых, при проектировании FDP много внимания уделялось упрощению системы ввода—вывода, которая, однако, оказалась неудобной для работы в реальном времени. В-пятых, объем памяти FDP ограничен архитектурой процессора. В-шестых, структура FDP малопригодна для обработки чисел с плавающей запятой, а также для выполнения программ численного анализа, требующих высокой точности.

При разработке процессора LSP2 была сделана попытка за счет использования более современных элементов устранить некоторые из перечисленных недостатков. Общая структурная схема процессора изображена на фиг. 11.14. Основой схемы является распределительная система, содержащая три шины. К ней подключены быстродействующее ЗУ небольшой емкости M_R, различные функциональные блоки и ЗУ большого объема M_S, связанное с внешними устройствами и ЗУ программы M_P.

Перечислим некоторые соображения, определившие структуру LSP2:

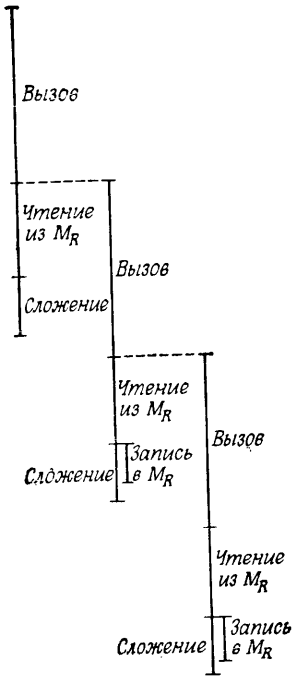
1. Быстродействующее ЗУ M_R вводится для того, чтобы освободить многие операции с индексами и обращения к памяти от вычислений, необходимых для этих операций. Для пояснения введем сначала формат команды обращения к ЗУ M_R:

КОП	A	B	D



Фиг. 11.14. Блок-схема процессора LSP2.

Она содержит адреса A, B и D трех полей памяти M_R, а также код КОП, определяющий нужный функциональный блок. По этой команде из двух полей ЗУ M_R, обозначенных через A и B, извлекаются два числа и по шинам A и B передаются на входы выбранного функционального блока. Результат (умножения, сложения и т. д.) поступает на шину D и записывается в ЗУ M_R по адресу D. Все это происходит за один командный цикл, длительность которого (в зависимости от исполняемой операции) составляет от 60 до 150 нс. На фиг. 11.15 схематично показано, как осуществляется управление при выполнении команд. Для примера рассмотрены три последовательных сложения. После вызова команды, занимающего 60 нс, за 30 нс числа извлекаются из памяти M_R и подаются на шины A и B. Для обеспечения такого быстродействия память M_R содержит два независимых ЗУ для адресов A и B, обращение к которым происходит одновременно. Вслед за этим производится собственно сложение, занимающее 20 нс. Результат операции возвращается в M_R не сразу: сначала он заносится во вспомогательный регистр, а записывается в M_R во время выполнения следующей команды. Такой прием позволяет сэкономить время, уходящее на запись в M_R. Трудностей с условными переходами не возникает, поскольку результат



Фиг. 11.15. Временна́я последовательность выполнения операций в процессоре LSP2.

сложения имеется во вспомогательном регистре, из которого поступают данные, требующиеся при вызове условного перехода.

Преимущества использования быстродействующей оперативной памяти M_R обусловлены тем, что с ее помощью можно обрабатывать значительные числовые массивы без обращения к основному ЗУ и даже без индексации. Хорошим подтверждением этому служит выполнение БПФ небольшого объема. Допустим, что M_R содержит 64 слова и нужно выполнить 32-точечное БПФ. После занесения всех 32 чисел в M_R программа БПФ может быть выполнена полностью без обращений к M_S и без переходов. Можно показать, что в этом случае БПФ вычисляется в LSP2 почти в шесть раз быстрее, чем в FDP. С учетом того, что LSP2 гораздо *проще* (но работает почти вдвое быстрее), такое улучшение архитектуры системы следует считать значительным шагом вперед.

2. Введение функциональных блоков связано с успехами в области создания быстродействующих микросхем. Это позволило ввести в процессор больше элементов, используемых только для арифметических операций, и не загружать их реализацией множества других функций. На первый взгляд такой подход кажется менее экономичным, но экономия на схемах управления, упрощение структуры и увеличение модульности процессора оправдывают его. Например, при наличии функционального блока с двумя умножителями для выполнения базовой операции БПФ требуется шесть команд. Если же имеется блок умножения комплексных чисел, то для получения того же результата достаточно трех команд. Аналогично нетрудно ввести и другие специализированные функциональные блоки. Так, некоторые образцы LSP2, предназначенные для ускорения решения некоторых частных задач, могут иметь специализированные блоки деления или извлечения квадратного корня. В большинстве ЦВМ добавление специализированных функциональных блоков связано с расходами на переходные (согласующие) устройства. В LSP2 такие блоки являются просто сменными узлами.

3. В LSP2 основным ЗУ является M_S. В зависимости от размеров машины адрес может состоять из 12 или 16 разрядов. С помощью специального переключения предусмотрено увеличение разрядности адресов с 12 до 16. Адресация ЗУ M_S может быть прямой в соответствии с кодом команды либо косвенной с учетом регистров M_R. Это означает, что M_R играет сразу две роли: служит памятью для обрабатываемых чисел и запоминает индексы, т. е. для этих действий не требуется параллельно работающих блоков. Но это также означает, что вычисления и операции с индексами нельзя проводить одновременно, как в FDP, поэтому для LSP2 могут потребоваться дополнительные команды. Однако в силу принципиально последовательной логики работы LSP2 программирование для него должно быть более простым.

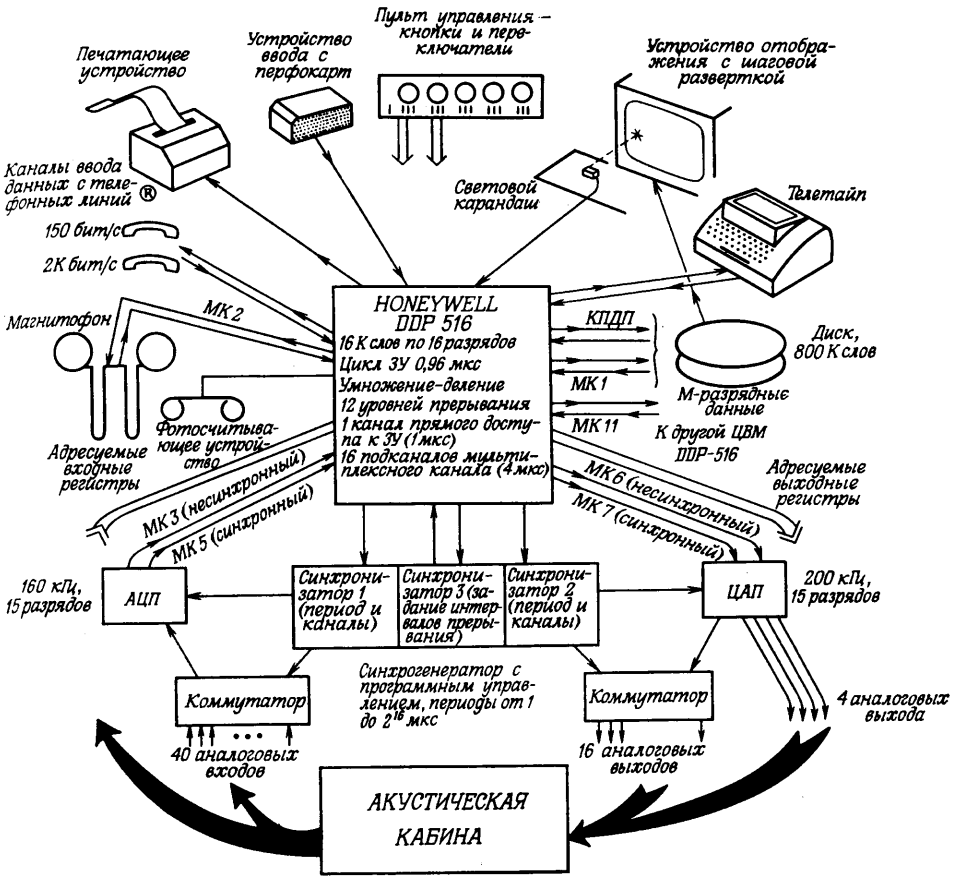
4. Связь машины с внешними устройствами осуществляется через память M_S. ЗУ имеет две входные и две выходные линии передачи данных, а также шесть управляющих (командных) линий, соединяющих его с другими устройствами. Поскольку многие операции выполняются с помощью M_R без привлечения M_S, а вводом и выводом управляют два отдельных устройства, LSP2 может весьма эффективно работать в реальном времени. Имеется в виду, что обе части программы (как вычислительная, так и осуществляющая ввод—вывод) выполняются без потерь времени. В частности, в LSP2 имеются возможности приоритетного прерывания и прямого доступа к ЗУ M_S, что позволяет объединять несколько процессоров для параллельной, причем весьма эффективной работы.

В настоящее время система LSP2 находится в стадии разработки и, чтобы оценить ее характеристики, необходимо накопить опыт работы с ней. Этот краткий обзор приведен здесь потому, что разработка LSP2 является естественным следствием опыта, накопленного при работе с FDP, а также продолжающегося развития техники быстродействующих микросхем.

11.18. Лабораторная вычислительная система для цифровой обработки сигналов

В настоящей главе в основном рассматривались способы ускорения обработки сигналов с помощью универсального оборудования (фактически быстродействующей ЦВМ, работающей с большими числовыми массивами), которое подключается к лабораторной универсальной мини-ЦВМ. Обычно бóльшая часть вычислений производится в быстродействующем процессоре, а на мини-ЦВМ с ее периферийным оборудованием возлагается задача обеспечения ввода—вывода всевозможных сигналов. В данном разделе рассматривается с точки зрения возможностей ввода—вывода лабораторная система, основанная на мини-ЦВМ.

На фиг. 11.16 представлена блок-схема вычислительного комплекса отдела акустики фирмы Bell Laboratories, скомпонованного на базе мини-ЦВМ типа Honeywell DDP-516. Эта вычислительная машина выполнена на интегральных схемах, имеет цикл обращения к памяти 0,96 мкс и оперирует с 16-разрядными словами. Объем ее оперативной памяти на магнитных сердечниках равен 16К. Имеются аппаратурные средства для умножения и деления, мультиплексный канал управления (МК) с 16 подканалами передачи данных (с быстродействием по 0,25 МГц), а также канал прямого доступа к памяти (КПДП) с пропускной способностью до 1 МГц. Математическое обеспечение машины включает транслятор с языка ФОРТРАН IV, Ассемблер (автокод), библиотеку стандартных программ, вводимую извне перемещаемую программу-загрузчик и различные вспомогательные программы типа программы отладки.



Фиг. 11.16. Вычислительная система для цифровой обработки сигналов.

Опыт показал, что при использовании мини-ЦВМ весьма желательно дополнить ее базовый комплект различными периферийными устройствами. В рассматриваемой системе к мини-ЦВМ подключены:

1. Два диска емкостью по 394К слов с фиксированными головками, временем доступа не более 33 мс и частотой обмена ~180 тыс. слов/с.

2. Четыре независимых канала преобразования цифровых сигналов в аналоговые с программным изменением частоты синхронизации в пределах от 0 до 180 кГц.

3. 15-разрядный аналого-цифровой преобразователь (АЦП) с частотой дискретизации 180 кГц и 40-канальный коммутатор с предельной частотой 100 кГц.

4. Устройство отображения с шаговой разверткой, управляемое с диска. Оно оказалось почти незаменимым при визуальном исследовании колебаний, спектров и т. д.

5. Устройство ввода с перфокарт и выдачи на перфокарты (для исправления программ вне ЦВМ) с быстродействием 300 карт/мин.

6. Печатающее устройство для распечатки результатов и отладочного материала.

7. Накопитель на магнитной ленте для хранения больших программ и массивов чисел.

8. Устройство считывания с перфолент для ввода программ, поставляемых изготовителем машины.

Операционная система — дисковая; она хранится на диске в защищенной от записи зоне объемом 48 К слов. Резидентная программа-загрузчик объемом в 15 слов, прошитая на магнитных сердечниках, обеспечивает удобный доступ к программам, хранящимся на диске. Весьма сложная система графического отображения позволяет использовать кинескоп с шаговой разверткой для отображения текста, колебаний и различной графической информации. Графические изображения результатов на бумаге могут быть получены через центральный процессор после предварительного вывода данных из оперативной памяти на магнитную ленту.

Как уже отмечалось, для эффективного использования быстродействующего процессора (подобного описанным в данной главе) необходимо иметь хорошую систему ввода—вывода, обеспечивающую отладку и работу с программами. В этом отношении разнообразные периферийные устройства, перечисленные выше, оказались весьма полезными.

ЛИТЕРАТУРА

1. Gold B., Lebow I. L., McHugh P. G., Rader C. M., The FDP, A Fast Programmable Signal Processor, *IEEE Trans. on Computers*, C-**20**, 33—38 (Jan. 1971).
2. Hornbuckle G. D., Ancona E. I., The LX-1 Microprocessor and its Application to Real-Time Signal Processing, *IEEE Trans. on Computers* (Aug. 1970).
3. Gschwind H. W., Design of Digital Computers, Springer-Verlag, Austria, 1967, pp. 235—243.
4. Blankenship P., Gold B., McHugh P., Weinstein C. J., Design Study of the Advanced Signal Processor, Lincoln Lab. Tech. Note, 1972-17, 1972.

ГЛАВА 12

ЦИФРОВАЯ ОБРАБОТКА РЕЧЕВЫХ СИГНАЛОВ

12.1. Введение

Одной из наиболее важных областей применения цифровых методов является обработка речевых сигналов. Фактически значительная часть теоретических результатов, составляющих основу цифровых методов обработки сигналов, была получена исследователями, изучавшими речевые сигналы. Ниже мы увидим, что цифровая обработка использовалась для решения широкого круга вопросов, включая спектральный анализ, полосные вокодеры, гомоморфные системы обработки, синтезаторы речи, системы линейного прогнозирования и системы голосового управления вычислительными машинами. В данной главе достаточно подробно рассмотрено несколько типичных речевых систем, при создании которых важную роль сыграла цифровая обработка. В главу включены примеры как аппаратурной, так и программной реализации систем обработки речи. Прежде чем перейти к конкретным примерам, будет дан обстоятельный обзор моделей образования речи.

12.2. Модель образования речи

На фиг. 12.1. изображена схема, описывающая механизм образования речи в человеческом организме. При разговоре грудная клетка расширяется и сжимается, прокачивая поток воздуха из легких по трахее через голосовую щель. Если голосовые связки напряжены, как при образовании звонких звуков типа гласных, то они вибрируют подобно релаксационному генератору и модулируют поток воздуха, превращая его в короткие импульсы (порции). Если голосовые связки расслаблены, воздух свободно проходит через голосовую щель, не подвергаясь модуляции. Воздушный поток проходит через глоточную полость мимо основания языка и в зависимости от положения мягкого нёба — через ротовую и (или) носовую полости. Поток воздуха выходит наружу через рот или нос (или обоими путями) и воспринимается как речь. В случае глухих звуков, таких, как *s* в слове snow или *p* в слове pit, голосовые связки расслаблены. При этом возможны два