Московский государственный технический университет
имени Н.Э. Баумана

Факультет «Информатика и управление»
Кафедра «Информационные системы и телекоммуникации»

В.С. Выхованец, Н.А. Демин, Е.И. Мозговая, С.И. Назарова, Д.А. Рожкова, Е.С. Шапкина

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА ОБРАБОТКИ СИГНАЛОВ

Электронное учебное издание

Учебное пособие по курсу
«Микропроцессорные устройства обработки сигналов»

Москва
© 2014 МГТУ им. Н.Э. Баумана

УДК 681.323

Рецензенты:

д.т.н., проф. Фархадов Маис Паша-Оглы

д.т.н., проф. Сюзев Владимир Васильевич

Микропроцессорные устройства обработки сигналов. Учебное пособие / Выхованец В.С., Демин Н.А., Мозговая Е.И., Назарова С.И., Рожкова Д.А., Шапкина Е.С.; Под ред. В.С. Выхованца. – М.: МГТУ им. Н.Э. Баумана, 2014. – 177 с.

Издание содержит сведения, необходимые для изучения дисциплины «Микропроцессорные устройства обработки сигналов». Основное внимание уделено связи аппаратурных и программных средств микропроцессорных устройств, предназначенных для обработки сигналов и изображений.

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлению 230400 – «Информационные системы и технологии».

Рекомендовано Научно-методическим советом МГТУ им. Н.Э. Баумана

Выхованец Валерий Святославович

Демин Никита Александрович

Мозговая Екатерина Игоревна

Назарова Светлана Игоревна

Рожкова Диана Александровна

Шапкина Евгения Сергеевна

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА ОБРАБОТКИ СИГНАЛОВ

© Выхованец В.С., Демин Н.А., Мозговая Е.И.,
Назарова С.И., Рожкова Д.А., Шапкина Е.С., 2014
© 2014 МГТУ им. Н.Э. Баумана

Оглавление

Предисловие

Настоящее учебное пособие предназначено для изучения дисциплины «Микропроцессорные устройства обработки сигналов» учебного плана МГТУ им. Н.Э. Баумана для подготовки магистров по направлению 230400 – «Информационные системы и технологии». Для успешного освоения материала необходимо предварительное изучение таких дисциплин как «Основы теории управления и цифровая обработка сигналов», «Микропроцессорная техника и цифровые автоматы», «Архитектура информационных систем», «Программирование на языке высокого уровня», «Разработка программного обеспечения».

Учебный материал разбит на два модуля. В первом модуле рассматривается архитектура и организация микропроцессоров, предназначенных для обработки сигналов и изображений. Второй модуль посвящен обработке данных и сигналов, где рассматриваются методы ускорения выполнения команд микропроцессором, а также задачи и методы эффективной цифровой обработки сигналов.

Каждый модуль содержит необходимые теоретические сведения, задание для домашней проработки учебного материала, постановку и порядок выполнения лабораторного исследования. По каждому домашнему заданию и лабораторной работе оформляется отчет. Конкретные требования к отчету приведены в описании домашних заданий и лабораторных работ.

По завершению изучения каждого модуля предусмотрен модульный контроль, основная цель которого – проверка качества усвоенного учебного материала. Для подготовки к модульному контролю в пособии содержатся контрольные вопросы по теоретической части материала, и учебные задачи – по практической части.

В пособии приведен обширный список литературы, который имеет цель указать работы, в которых вопросы, кратко упомянутые в пособие, изложены подробно. Учитывая учебное назначение издания, многочисленные ссылки на использованную литературу были опущены.

Модуль 1
Архитектура и организация микропроцессоров

1.1 Микропроцессоры для обработки сигналов

Рассмотрим архитектуру и организацию микропроцессоров для цифровой обработки сигналов на примере микропроцессора TMS320C5515™ компании Texas Instrument®. Основные характеристики микропроцессора TMS320C5515™:

- корпус PBGA размером 10 на 10 мм и высотой 1,3 мм, 196 контактов;
- технологический процесс изготовления с разрешением 0,09 мкм;
- тактовая частота 60, 75, 100 и 120 МГц;
- время цикла 8,33, 10, 13,3 и 16,67 нс;
- напряжение питания ядра 1,05 и 1,30 В;
- напряжение питания периферийных устройств 1,8, 2,5, 2,75 и 3,3 В;
- мощность потребления в режиме энергосбережения от 0,15 до 0,28 мВт;
- мощность потребления в режиме работы от 18,0 до 26,4 мВт;
- быстродействие до 240 миллионов команд умножения со сложением в секунду;
- конвейерная выборка, декодирование и выполнение команд (12 стадий);
- параллельное выполнение двух команд;
- данные с фиксированной запятой разрядности 16, 32 и 40 бит;
- встроенная основная память объемом 320 Кб;
- встроенная постоянная память объемом 128 Кб;
- внешняя синхронная и асинхронная память объемом до 16 Мб.

На рисунке 1.1 показана структурная схема микропроцессора, включающая следующие блоки:

– ядро микропроцессора (DSP Core) с аппаратурным ускорителем быстрого преобразования Фурье (FFT);

– внутренняя память, состоящая из постоянного запоминающего устройства (ROM), основной двухвходовой памяти (DARAM) и основной одновходовой памяти (SARAM);

– четыре контроллера прямого доступа к памяти (DMA), каждый с четырьмя независимыми каналами;

– интерфейс внешней памяти (EMIF) с 21-разрядной шиной адреса и 16-разрядной магистралью данных, к которому может быть подключена мобильная синхронная динамическая память (mSDRAM) и асинхронная флэш-память на элементах НЕ-И (NAND);

– тактовый генератор с устройством фазовой автоподстройки частоты (PLL), на вход которого подключается часовой кварцевый резонатор с частотой 32,768 кГц;

– периферийные устройства: входы-выходы общего назначения (GPIO), устройства чтения-записи мультимедийных карт (MMC/SD), звуковые интерфейсы (I2S), приборный интерфейс (I2C), часы реального времени (RTC), таймеры (Timer), универсальный последовательный интерфейс (USB 2.0), универсальный асинхронный приемо-передатчик (UART), высокоскоростной последовательный интерфейс (SPI), контроллер жидкокристаллического дисплея (LCD), аналогово-цифровой преобразователь (SAR);

– три независимых стабилизированных источника (LDO) питания: для цифрового питания ядра микропроцессора и периферийных устройств, для аналогового питания периферийных устройств и для цифрового питания универсального последовательного интерфейса USB 2.0;

– контроллер прерываний (INT);

– эмулятор для отладки программ (JTAG).

1.1.1 Ядро микропроцессора

Ядро микропроцессора выполняет цифровую обработку сигналов и действует как контроллер, ответственный за реализацию системных функций, таких как начальная инициализация и конфигурация устройств микропроцессора, организация взаимодействия между устройствами и т.п. Непосредственно с ядром микропроцессора связаны следующие компоненты (рисунок 1.1):

– двухвходовая память (DARAM);

– одновходовая память (SARAM);

– постоянная память (ROM);

– аппаратурный ускоритель для быстрого преобразования Фурье (FFT);

– внутренняя шина периферийных устройств;

– шина для подключения внешней памяти (EMIF).

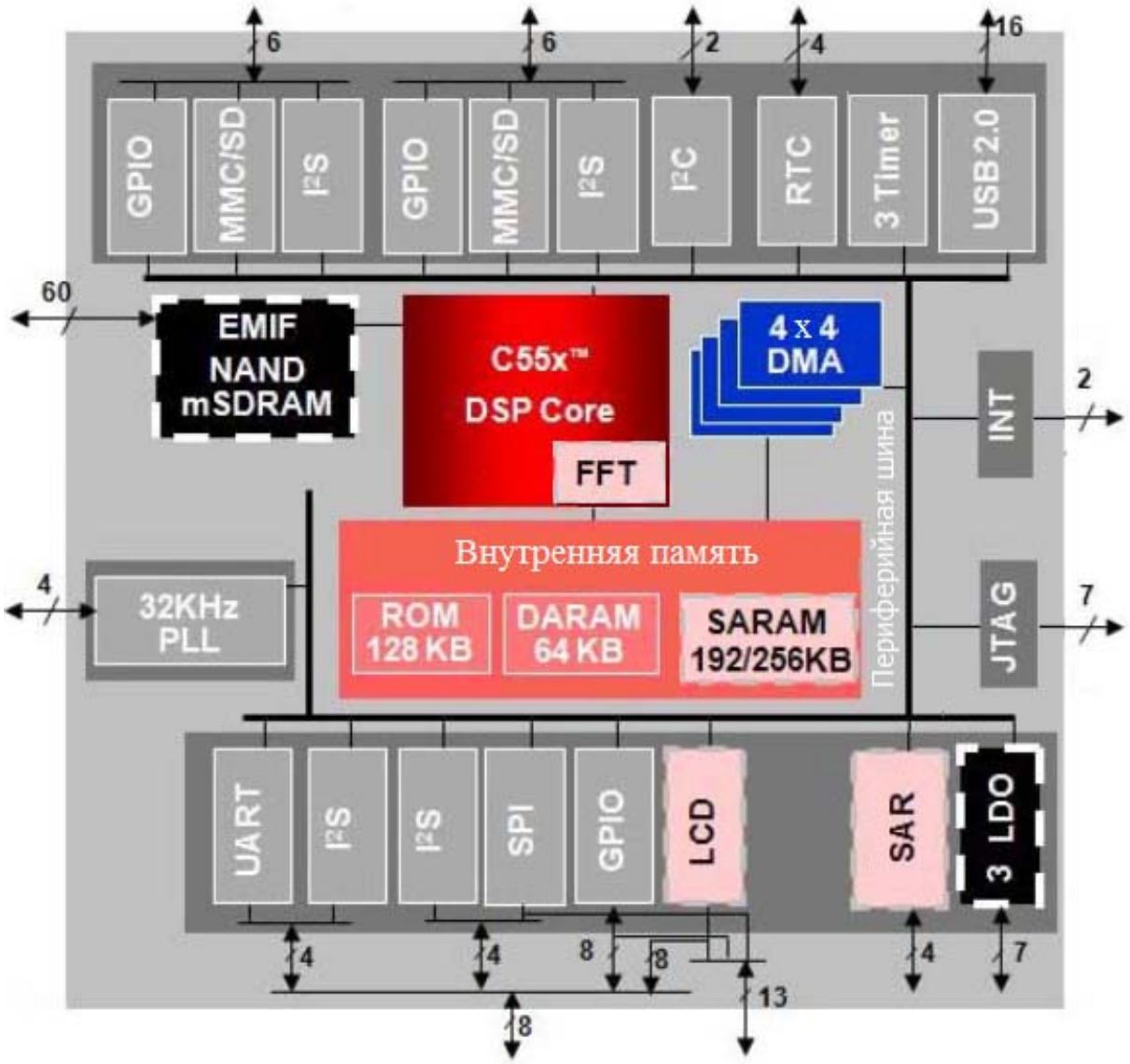


Рисунок 1.1 – Структурная схема микропроцессора

Состав устройств, входящих в ядро микропроцессора, показан на рисунке 1.2. Основными структурными частями ядра являются шины адреса и магистрали данных для чтения команд и чтения-записи данных, процессор, интерфейс внутренней и внешней памяти. Шины адреса и магистрали данных ядра микропроцессора перечислены в таблице 1.1.

Магистраль PB (Program Bus) служит для переноса 32-разрядных данных из внутренней памяти программ в буфер команд (I модуль), где команды, содержащиеся в поступивших данных, декодируются и передаются для исполнения в функциональные устройства ядра микропроцессора: устройство управления (P модуль), устройство адресации (A модуль) и операционное устройство (D модуль). Управляет последовательной выборкой команд из памяти устройство управления, формируя на

шине адреса PAB (Program address bus) последовательность 24-разрядных адресов 8-разрядных байт.

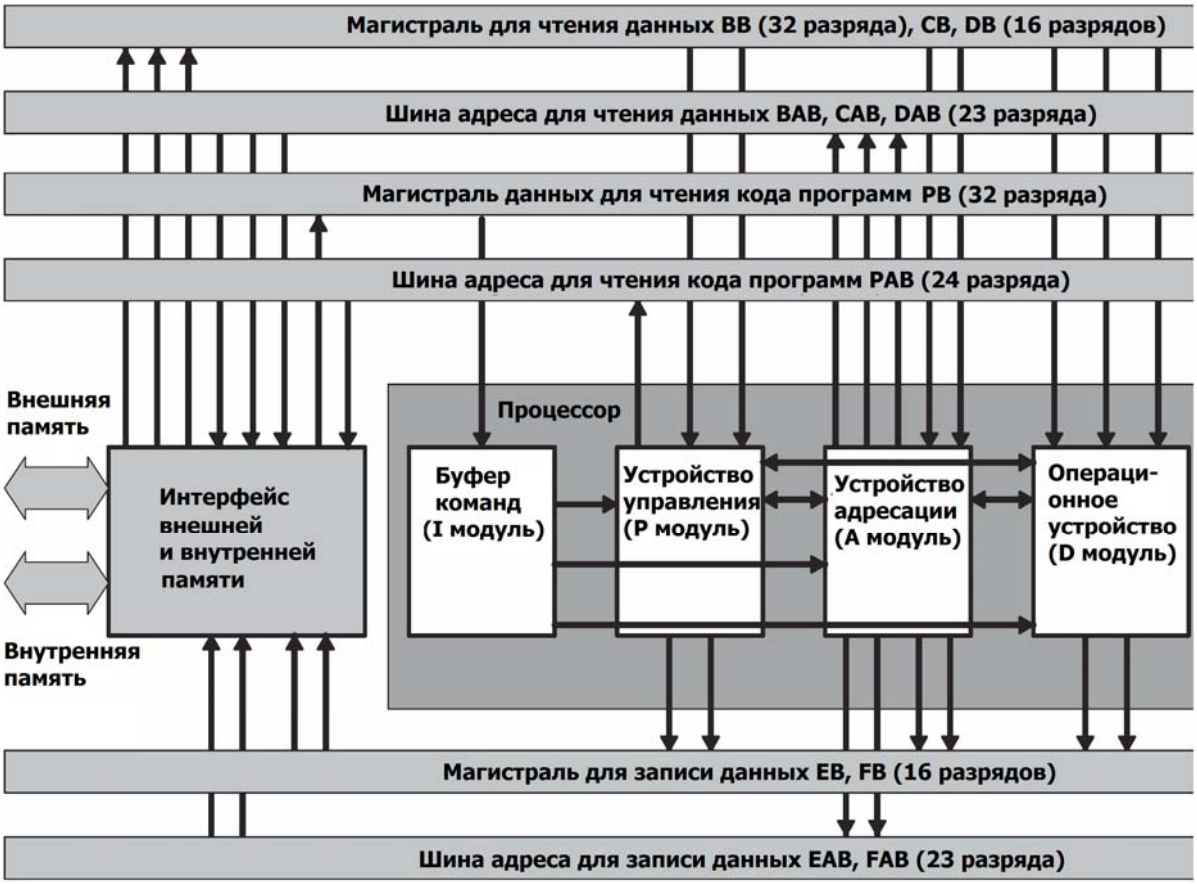


Рисунок 1.2 – Ядро микропроцессора

Таблица 1.1 – Шины адреса и магистрали данных ядра микропроцессора

Шина, магистраль	Разрядность	Назначение
PAB	24	Шина адреса для чтения 32-разрядного кода программ
PB	32	Магистраль для чтения 32-разрядного кода программ
CAB, DAB	23	Шины адреса для чтения 16-разрядных данных
CB, DB	16	Магистрали для чтения 16-разрядных данных
BAB	23	Шина адреса для чтения 32-разрядных данных
BB	32	Магистраль для чтения 32-разрядных данных
EAB, FAB	23	Шины адреса для записи 16-разрядных данных
EB, FB	16	Магистрали данных для записи 16-разрядных данных

Магистраль чтения данных BB (англ. B bus) переносит 32-разрядные, а магистрали CB (англ. C bus) и DB (англ. D bus) – 16-разрядные данные. Перенос осуществляется к функциональным модулям ядра микропроцессора из пространств внешней и внутренней памяти или внутреннего пространства ввода-вывода, в котором размещены регистры периферийных устройств. В свою очередь магистрали EB (англ. E bus) и FB (англ. F bus) переносят данные от функциональных модулей ядра микропроцессора в адресные пространства памяти и ввода-вывода.

Магистраль BB соединяет операционное устройство только с внутренней памятью микропроцессора, поэтому 32-разрядные операнды операционного устройства не могут размещаться во внешней памяти, так же как не может иметь 32-разрядных операндов, размещаемых в памяти, устройство управления и устройство адресации.

Дополнительные 16 разрядные операнды могут поступать в операционное устройство, устройство адресации или устройство управления по магистралям CB и DB одновременно с 32-разрядным операндом, поступающим по магистрали BB. В отличие от магистрали BB магистрали CB и DB переносят данные не только из внутренней, но и из внешней памяти микропроцессора.

Таким образом, команды операционного устройства могут иметь до трех операндов-источников и до двух операндов-приемников данных:

– один 32-разрядный операнд-источник, поступающий по магистрали BB из внутренней памяти;

– два 16-разрядных операнда-источника, или один 32-разрядный операнд-источник, поступающих по магистралям CB и DB из внутренней или внешней памяти;

– два 16-разрядных операнда-приемника, или один 32-разрядный операнд-приемник, передаваемые по магистралям EB и FB во внутреннюю или внешнюю память.

В свою очередь команды, реализуемые устройством управления и устройством адресации, могут иметь не более двух операндов-источников и не более двух операндов-приемников:

– два 16-разрядных операнда-источника, или один 32-разрядный операнд-источник, поступающих по магистралям CB и DB;

– два 16-разрядных операнда-приемника, или один 32-разрядный операнд-приемник, передаваемые по магистралям EB и FB.

Команды с двумя 16-разрядными операндами-источниками из памяти используют магистрали CB и DB, а команды с одним 16-разрядным операндом-источником –

только магистраль DB. В свою очередь команды с двумя 16-разрядными операндами-приемниками из памяти используют магистрали EB и FB, а с одним – магистраль EB. В качестве операндов-источников и операндов-приемников данных могут служить также регистры устройства управления, устройства адресации и операционного устройства, поступающие по двунаправленным шинам на рисунке 1.2.

Шинами адреса чтения данных по магистралям BB, CB, DB и шинами адреса записи данных по магистралям EB и FB являются шины BAB (англ. B Address Bus), CAB (англ. C Address Bus), DAB (англ. D Address Bus), EAB (англ. E Address Bus) и FAB (англ. F Address Bus) соответственно. Эти шины переносят 23-разрядные адреса читаемых и записываемых 16-разрядных слов от устройства адресации к интерфейсу внешней и внутренней памяти.

Циклы чтения и записи, выполняемые ядром микропроцессора, перечислены в таблице 1.2.

Таблица 1.2 – Циклы чтения и записи

Цикл	Шины, магистрали	Назначение
Выборка команды	PAB, PB	32-разрядное чтение из адресного пространства памяти программ
Короткое чтение данных	DAB, DB	16-разрядное чтение регистра, памяти, пространства ввода-вывода
Короткая запись данных	EAB, EB	16-разрядная запись в регистр, в память или в пространство ввода-вывода
Длинное чтение данных	DAB, (CB, DB)	32-разрядное чтение регистра или памяти
Длинная запись данных	EAB, (EB, FB)	32-разрядная запись в регистр или память
Двойное чтение данных	DAB, DB; CAB, CB	Два 16-разрядных чтения регистра (памяти, ввода-вывода) или памяти
Двойная запись данных	EAB, EB; FAB, FB	Две 16-разрядных записи регистра (памяти, ввода-вывода) или памяти
Короткое чтение и короткая запись данных	DAB, DB; EAB, EB	16-разрядное чтение памяти и 16-разрядная запись в память

Таблица 1.2 – Циклы чтения и записи

Цикл	Шины, магистрали	Назначение
Длинное чтение и длинная запись данных	DAB, (CB, DB); EAB, (EB, FB)	32-разрядное чтение памяти и 32-разрядная запись в память
Короткое чтение памяти и коэффициентов.	DAB, DB; BAB, BB	16-разрядное чтение памяти и 16- разрядное чтение коэффициентов
Короткое чтение памяти и длинное чтение коэф.	DAB, DB; BAB, BB	16-разрядное чтение памяти и 32- разрядное чтение коэффициентов
Двойное чтение памяти и короткое чтение коэффициентов	DAB, DB; CAB, CB; BAB, BB	Два 16-разрядных чтения память и 16- разрядное чтение коэффициентов
Двойное чтение памяти и длинное чтение коэффициентов	DAB, DB; CAB, CB; BAB, BB	Два 16-разрядных чтения памяти и 32- разрядное чтение коэффициентов

1.1.2 Адресные пространства

Микропроцессор имеет два адресных пространства: адресное пространство памяти и адресное пространство ввода-вывода, формирование которых осуществляет интерфейс внутренней и внешней памяти ядра микропроцессора (рисунок 1.2). В свою очередь адресное пространство памяти делится на адресное пространство внутренней и внешней памяти, а адресное пространство ввода-вывода является только внутренним.

Общий объем адресного пространства памяти микропроцессора – 16 Мб (рисунок 1.3). Микропроцессор не поддерживает раздельные адресные пространства для хранения программ и для чтения-записи данных, т.е. области памяти с кодом программ и области памяти с данными могут чередоваться произвольным образом и размещаться как во внутренней памяти, так во внешней памяти.

Внутренняя память микропроцессора конструктивно размещается в его корпусе и состоит из устройства памяти с произвольным доступом (RAM) объемом 320 кБ и памяти постоянного запоминающего устройства (ROM) объемом 128 кБ. В свою очередь внутренняя память с произвольным доступом разделяется:

– на регистровую память (англ. MMR, Memory Mapped Registers), через которую может осуществляться доступ к регистрам ядра микропроцессора в диапазоне адресов 000000h – 0000BFh (см. приложение А);

– на двухвходовую память (DARAM) к которой за один цикл доступа возможно одновременные обращения по двум произвольным адресам из диапазона адресов 0000C0h − 00FFFFh;

– на одновходовую память (SARAM) к которой за один цикл доступа возможно обращение только по одному адресу из диапазона адресов 010000h – 04FFFFh.

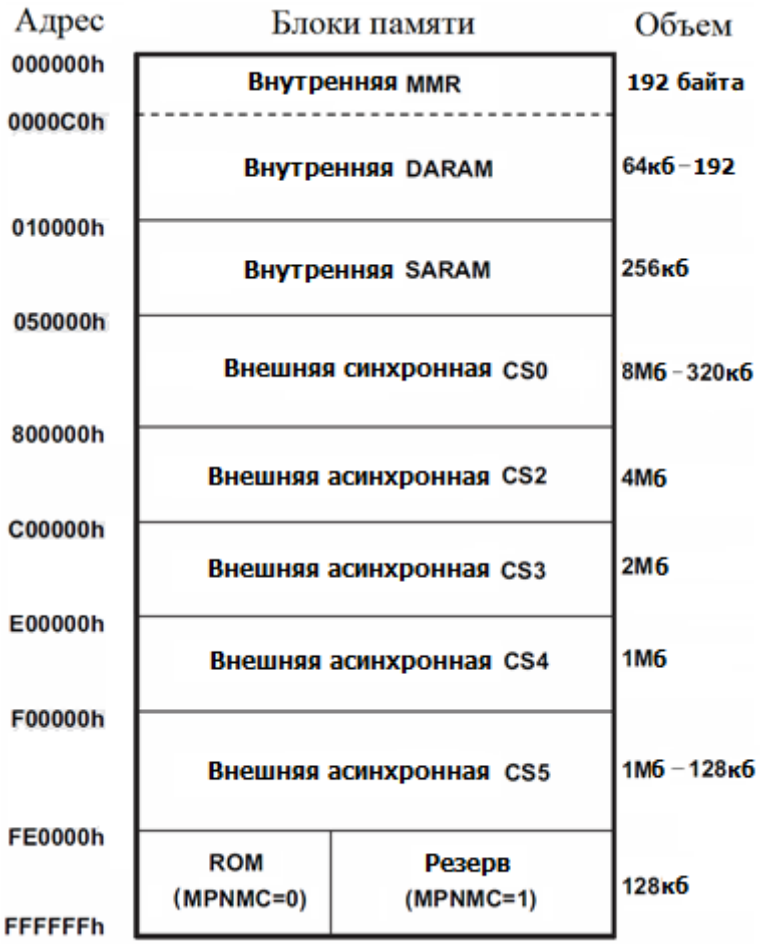


Рисунок 1.3 – Адресное пространство памяти

Постоянное запоминающее устройство предназначено для хранения программ и неизменяемых данных, необходимых для функционирования микропроцессора, а именно:

– программы сброса и инициализации микропроцессора;

– загрузчика программ из внешней памяти или из запоминающих устройств, подключенных через интерфейсы SPI и I2C;

– подпрограмм для выполнения прямого и обратного быстрого преобразования Фурье;

– начальной таблицы векторов прерываний (см. пп. 1.1.7.7);

– табличных данных для вычисления математических функций.

В зависимости от состояния флага MPNMC в регистре статуса микропроцессора ST3 (см. пп. 1.1.7.6) постоянное запоминающее устройство может подключаться к адресному пространству памяти или отключаться от него. В последнем случае диапазон адресов FE0000h – FFFFFFh, выделенный для постоянного запоминающего устройства, отображается во внешнюю память, что позволяет конструктивно размещать постоянное запоминающее устройство вне корпуса микропроцессора.

Внешняя память размещается в диапазоне адресов 050000h – FFFFFFh и делится на шесть подпространств:

– два подпространства CS0 и CS1 общим объемом до 8 Мб для размещения устройств синхронной динамической памяти (SDRAM), выбираемые соответственно при активных сигналах EM_CS0 или EM_CS1 на выводах микропроцессора;

– четыре подпространства CS2, CS3, CS4 и CS5 общим объемом до 8 Мб для размещения устройств асинхронной памяти (статических запоминающих устройств произвольной выборки данных, флэш-памяти типа NOR или NAND, и т.п.), выбираемых соответственно при активных сигналах на выводах микропроцессора EM_CS2, EM_CS3, EM_CS4 и EM_CS5.

Контроллер интерфейса внешней памяти EMIF (рисунок 1.1) управляет обменом данными с внешними запоминающими устройствами и через свои регистры, доступные через адресное пространство ввода-вывода, настраивается на их временные характеристики и разрядность (8 или 16 разрядов данных).

Адресное пространство ввода-вывода имеет объем 128 кб (65536 слов) и служит для чтения и записи регистров управления микропроцессором и регистров периферийных устройств (рисунок 1.4).

Доступ в адресное пространство ввода-вывода осуществляется с помощью специальных команд микропроцессора, а также каналами прямого доступа к памяти, выполняющими пересылку данных между регистрами периферийных устройств и областями адресного пространство памяти.

Адрес	Регистры периферийных устройств
0000h	Управление простоем (Idle Control)
0005h	Зарезервировано
0C00h	Канал прямого доступа DMA0
0C80h	Зарезервировано
0D00h	Канал прямого доступа DMA1
0D80h	Зарезервировано
0E00h	Канал прямого доступа DMA2
0E80h	Зарезервировано
0F00h	Канал прямого доступа DMA3
0F80h	Зарезервировано
1000h	Интерфейс внешней памяти EMIF
10EEh	Зарезервировано
1800h	Таймер Timer0
1820h	Зарезервировано
1840h	Таймер Timer1
1860h	Зарезервировано
1880h	Таймер Timer2
1900h	Часы реального времени RTC
1980h	Зарезервировано
1A00h	Контроллер I2C
1A6Dh	Зарезервировано
1B00h	Контроллер UART
1B80h	Зарезервировано
1C00h	Управление микропроцессором
1D00h	Зарезервировано
2800h	Контроллер I2S0
2900h	Контроллер I2S1
2A00h	Контроллер I2S2
2B00h	Контроллер I2S3
2C41h	Зарезервировано
2E00h	Контроллер LCD
2E41h	Зарезервировано
3000h	Контроллер SPI
3010h	Зарезервировано
3A00h	Контроллер MMC/SD0
3A20h	Зарезервировано
3B00h	Контроллер MMC/SD1
3B2Fh	Зарезервировано
7000h	Аналого-цифровой преобразователь SAR
7100h	Зарезервировано
8000h FFFFh	Контроллер USB

Рисунок 1.4 – Адресное пространство ввода-вывода

С целью снижения энергопотребления периферийные устройства могут отключаться путем установки специальных разрядов (флагов) в регистрах управления микропроцессором. Если периферийное устройство отключено, то его регистры становятся недоступными и не отображаются в адресное пространство ввода-вывода.

1.1.3 Представление чисел

Микропроцессор выполняет операции над числами, представленными в целочисленном формате со знаком или без знака, а также в формате с фиксированной запятой.

1.1.3.1 Форматы чисел

Целые числа представляются в виде последовательности битов (нулей и единиц) фиксированного размера, кратного 16 битам. Для кодирования целых чисел используются 16-разрядные слова и 32-разрядные двойные слова.

Формат N16 имеет разрядность 16 бит и используется для представления с единичной точностью чисел без знака в диапазоне от 0 до 65536, а формат Z16 – для представления чисел со знаком в диапазоне от -32768 до 32767 (рисунок 1.5).

Все биты числа принимают значение ноль или единица и нумеруются начиная с нуля. Номер бита в слове (двойном слове) называется разрядом. Каждому разряду присваивается вес. Бит в нулевом разряде считается самым младшим и имеет наименьший вес. Значение числа определяется как сумма весов тех разрядов, в которых значение бита равно единице.

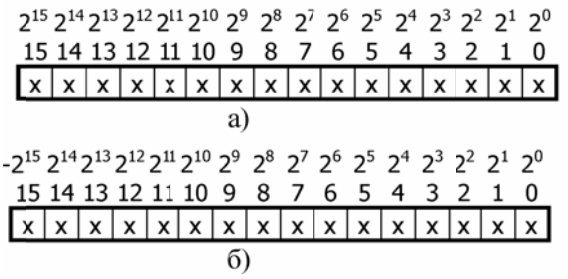


Рисунок 1.5 – Целочисленные форматы одинарной точности:
а) без знака N16, б) со знаком Z16

Например, последовательность бит 1000000000001001b (8009h) в формате N16 кодирует число, равное $2^{15}+2^3+2^0$, или 32777 в десятичной системе счисления, а в формате Z16 – число, равное $-2^{15}+2^3+2^0$, или -32757. В свою очередь последовательность бит 0000000000001001b (0009h) в формате N16 и Z16 кодирует одно и то же число, равное 2^3+2^0.

Формат N32 имеет разрядность 32 бита и используется для представления с единичной точностью чисел без знака в диапазоне от 0 до 4294967296, а формат Z32 – для представления чисел со знаком в диапазоне от -2147483648 до 2147483647 (рисунок 1.6).

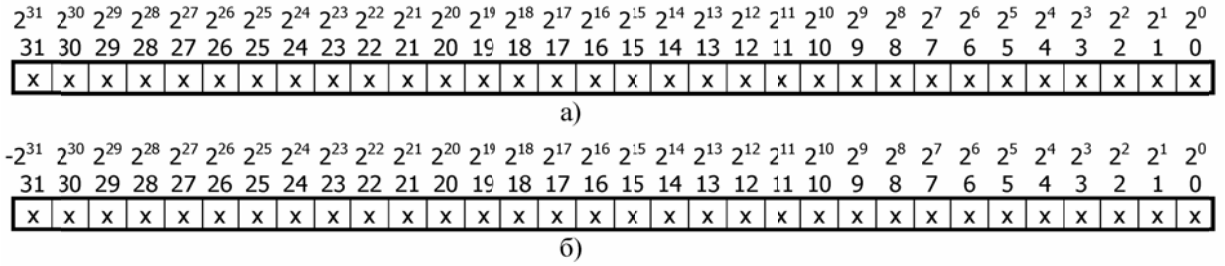


Рисунок 1.6 – Целочисленные форматы двойной точности:

а) без знака N32, б) со знаком Z32

Например, последовательность бит 10000000000000000000000000001001b (80000009h) в формате N32 кодирует число, равное $2^{31}+2^{3}+2^{0}$, или 2147483657, а в формате Z32 – число, равное $-2^{31}+2^{3}+2^{0}$, или -2147483639.

Дробные числа, так же как и целые, представляются последовательностью битов фиксированной длины, однако имеют другие веса разрядов. Для кодирования дробных чисел используются форматы Q4.12, Q1.15 и Q1.31.

Формат чисел с фиксированной запятой Q4.12 имеет разрядность 16 бит и используется для представления дробных чисел в диапазоне от -8 до 8 с точностью 2^{-12}, или $2{,}44\times10^{-4}$ (рисунок 1.7).

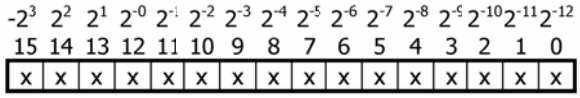


Рисунок 1.7 – Формат Q4.12

Формат чисел с фиксированной запятой Q1.15 также имеет разрядность 16 бит, но используется для представления дробных чисел в диапазоне от -1 до 1 с точностью 2^{-15}, или $3{,}05\times10^{-5}$ (рисунок 1.8).

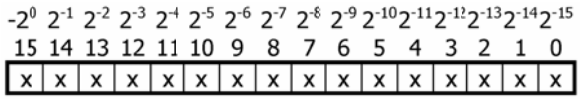


Рисунок 1.8 – Формат Q1.15

Например, последовательность бит 1000000000001001b (8009h) в формате Q4.12 кодирует число, равное $-2^{3}+2^{-9}+2^{-12}$, или -7,997802734375, а в формате Q1.15 – число, равное $-2^{0}+2^{-12}+2^{-15}$, или -0,999725341796875. В свою очередь последовательность бит 0000000000001001b (00000009h) в формате Q4.12 равна $2^{-9}+2^{-12}$, или 0,002197265625, а в формате Q1.15 – $2^{-12}+2^{-15}$, или 0,000274658203125.

В свою очередь формат чисел с фиксированной запятой Q1.31 имеет разрядность 32 бита, также используется для представления дробных чисел в диапазоне от -1 до 1, но уже с точностью 2^{-31}, или $4{,}66\times10^{-10}$ (рисунок 1.9).

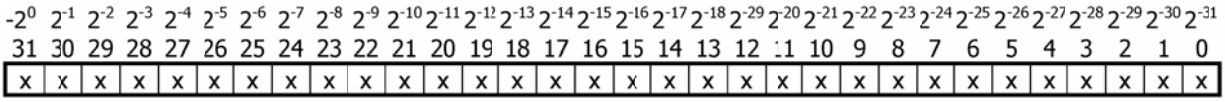


Рисунок 1.9 – Формат Q1.31

Например, последовательность бит 10000000000000000000000000001001b (80000009h) в формате Q1.31 кодирует число, которое равно $-2^0+2^{-28}+2^{-31}$, или -0,999999958090484142303466796875, а последовательность бит 00000000000000000000000000001001b (00000009h) – число, равное $2^{-28}+2^{-31}$, или 0,000000004190951585769653320312 5.

1.1.3.2 Хранение чисел в памяти

Минимальной единицей памяти, доступной микропроцессору для чтения или записи, является 16-разрядное слово. Поэтому для хранения 16-разрядных и 32-разрядных чисел в памяти выделяется одна (слово) или две смежные (двойное слово) 16-разрядные ячейки памяти соответственно.

Однако порядок слов в многословных числах может быть разным (рисунок 1.10):

– число записывается в порядке увеличения адресов, начиная с младшего слова и заканчивая старшим (от младшего к старшему, англ. little-endian);

– число записывается в порядке увеличения адресов, начиная со старшего слова и заканчивая младшим (от старшего к младшему, англ. big-endian).

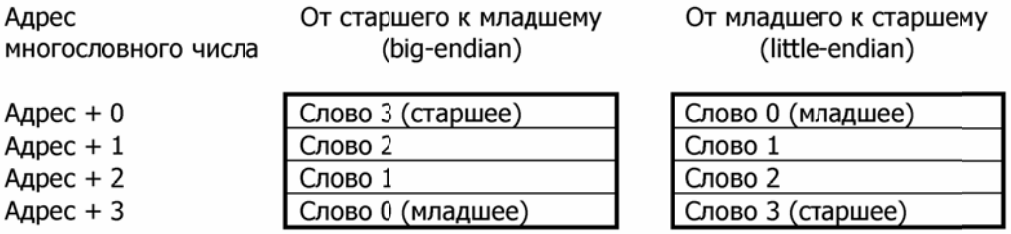


Рисунок 1.10 – Порядок слов в многословных числах

Микропроцессор реализует второй способ хранения многосложных слов, а именно, от старшего слова числа к младшему, при котором младшее слово хранится в ячейке с большим адресом, а старшее слово – в следующей ячейке с меньшим адресом.

Дробные числа хранятся в памяти в таких же ячейках памяти, как и целые. Поэтому 16-разрядная ячейка (слово) может быть проинтерпретирована как хранящая целое число со знаком или без знака, а также дробное число в формате Q4.12 или в

Q1.15. Ячейка памяти с разрядностью 32 бита (двойное слово) также может интерпретироваться несколькими способами: как целое число со знаком, как целое число без знака или как дробное число в формате Q1.31.

1.1.3.3 Преобразование форматов

Перед выполнением операций над целыми и дробными числами последние могут преобразовываться микропроцессором во внутренний 40-разрядный формат. Представление чисел в 40-разрядных форматах показано на рисунке 1.11.

Преобразование 32-разрядных дробных и целых чисел со знаком в 40-разрядный формат осуществляется копированием 32 разрядов числа в младшие 32 разряда 40-разрядного представления и расширением знака путем копирования 31 разряда числа в разряды 32-39. Целые без знака преобразовываются аналогично, но знаковый разряд не копируется, а разряды 32-39 просто обнуляются.

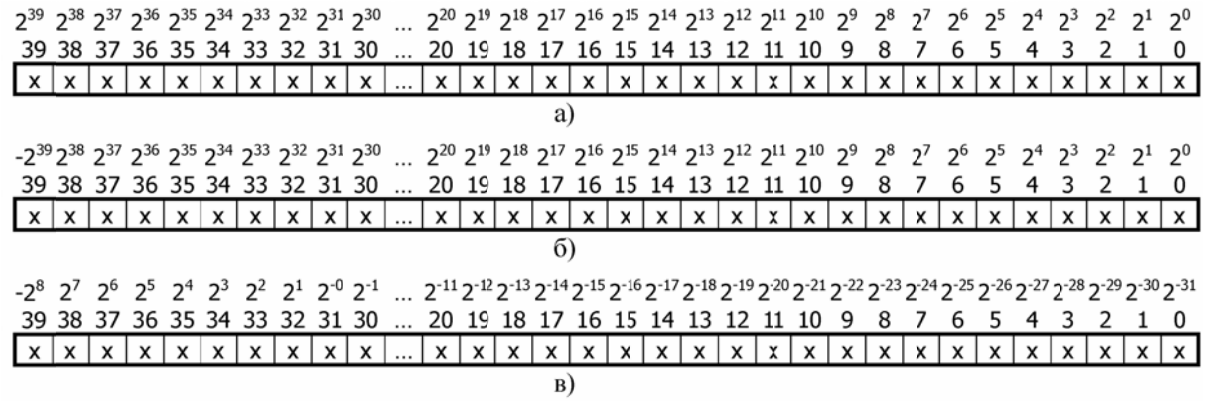


Рисунок 1.11 – Расширенные форматы чисел:
а) целое без знака N40, б) целое со знаком Z40, в) дробное Q9.31

Например, 32-разрядное число 80000009h, интерпретируемое как дробное или целое со знаком, преобразуется в 40-разрядное число FF80000009h, а то же число, но интерпретируемое как целое без знака – в число 0080000009h. В обоих случаях значения исходного и результирующих чисел будут равны.

Преобразование 40-разрядных чисел в 32-разрядные осуществляется простым копированием младших 32 разрядов. При таком преобразовании знаковых форматов искажение результата не будет тогда и только тогда, когда старшие 8 разрядов исходного 40-разрядного числа равны знаковому разряду результата. В свою очередь при беззнаковом кодировании чисел преобразование сохранит значение числа, если старшие 8 разрядов равны нулю.

Например, независимо от формата представления 40-разрядное число 0180000009h преобразуется в 32-разрядное число 80000009h. Однако при этом

наблюдается искажение результата, так как значение исходного 40-разрядного числа не равно значению результирующего.

Преобразование 16-разрядных дробных чисел в 40-разрядный формат осуществляется аналогично, но перед выполнением операции 16-разрядное число преобразуется в 32-разрядное путем добавления 16 нулевых младших разрядов.

Например, 16-разрядное дробное число 8009h в формате Q1.15 преобразуется в 32-разрядное число 80090000h в формате Q1.31, сохраняя при этом свое значение.

Преобразование 40-разрядных дробных чисел в 16-разрядные осуществляется простым копированием разрядов с 16 по 31. Очевидно, что в этом случае происходит потеря точности представления числа.

Например, 40-разрядное дробное число FF80000009h преобразуется в 16-разрядное число 8000h. В результате такого преобразования значение исходного числа искажается, однако полученный результат является наиболее близким к исходному по своему значению.

Преобразование 32-разрядного дробного числа в 16-разрядное дробное осуществляется путем копирования его старших 16-разрядов, а преобразование 32-разрядных целых чисел в 16-разрядные – путем копирования младших 16 разрядов.

Например, 32-разрядное дробное число 80000009h преобразуется в 16-разрядное число 8000h, а то же число, рассматриваемое как целое, преобразуется в 0009h. В первом случае полученное значение числа наиболее близко к исходному, а во втором случае наблюдается значительное искажение результата.

1.1.3.4 Насыщение

При преобразовании 40-разрядных дробных чисел в дробные числа с меньшей разрядностью возможна ситуация, при которой значение исходного числа не совпадает со значение результирующего. Такая же ситуация возможна и при выполнении некоторых других операций над числами.

Для уменьшения отрицательных последствий, связанных с выходом чисел за пределы диапазона представления, микропроцессор может выполнять корректировку результата. Если результат какой-либо операции становится непредставимым в используемом формате, то микропроцессор изменяет этот результат таким образом, чтобы заменить значение, вышедшее за диапазон представления наиболее близким к нему, но представимым значением. Такая операция называется насыщением (saturation).

Например, при преобразовании 40-разрядных дробных чисел FE80000009h и 0180000009h, заданных в формате Q9.31 в числа в формате Q1.31 без корректировки получим один и тот же результат 800009h, или -0,99999999580904841423034667968 75, хотя первое число равно -2,99999999580904841423034667968 75, а второе – +3,00000000419095158576965332031 25. При включенном режиме насыщения микропроцессор перед преобразованием заменит первое число самым маленьким числом из диапазона Q1.31 (FF80000000h, или -1), а второе – самым большим (007FFFFFFFh, или 0,99999999953433871269226074218 75). В итоге преобразования будет получен следующий результат: 80000000h и 7FFFFFFFh.

1.1.3.5 Округление

Округление – операция над числом, позволяющая уменьшить число цифр за счёт замены точного значения числа на его приближённое значение, вычисляемое с заданной точностью. Во всех методах округления лишние двоичные цифры отбрасываются (обнуляются), а предшествующая им цифра корректируется по какому-либо правилу.

Известны 5 видов округления дробных чисел:

– округление к ближайшему числу (англ. rounding);

– округление к меньшему числу (англ. floor);

– округление к большему числу (англ. ceiling);

– округление к меньшему по модулю числу (англ. fix, truncate);

– округление к большему по модулю числу (англ. integer).

В случае если отбрасываемая цифра равна 5 (1 в двоичной системе счисления) и последующие цифры равны нулю, то для округления безразлично, производить его к меньшему или к большему числу – в обоих случаях вносится погрешность, равная половине младшего разряда.

Однако когда происходит сложение большого числа таких округленных значений, как это имеет место при цифровой обработке сигналов, использование округления только к меньшему или только к большему числу вносит систематическую ошибку. В этом случае используются другие варианты округления:

– математическое округление – округление всегда в большую по модулю сторону (предыдущий разряд всегда увеличивается на единицу), то есть 0,5 ≈ 1, 1,5 ≈ 2, 2,5 ≈ 3, 3,5 ≈ 4, 4,5 ≈ 5;

– банковское округление – округление происходит к ближайшему чётному числу, то есть 0,5 ≈ 0, 1,5 ≈ 2, 2,5 ≈ 2, 3,5 ≈ 4, 4,5 ≈ 4;

– случайное округление – округление происходит в меньшую или большую сторону в случайном порядке, но с равной вероятностью (может использоваться в статистике), например: 0,5 ≈ 0, 1,5 ≈ 1, 2,5 ≈ 3, 3,5 ≈ 4, 4,5 ≈ 4;

– чередующееся округление – округление происходит в меньшую или большую сторону поочерёдно, то есть 0,5 ≈ 1, 1,5 ≈ 1, 2,5 ≈ 3, 3,5 ≈ 3, 4,5 ≈ 5.

В цифровой обработке сигналов используется математическое и банковское округление. Но только при использовании банковского округления исчезает систематическая ошибка при суммировании большого числа округленных значений. Поэтому все числа, у которых отбрасываемая цифра не равна 5 (1 в двоичной системе счисления), округляются по правилам математического метода. А другие числа – по правилу банковского округления:

– если цифра, которая стоит перед цифрой 5 (1 в двоичной системе счисления), четная (0 в двоичной системе счисления), то округление осуществляется в меньшую сторону;

– если цифра, которая стоит перед цифрой 5 (1 в двоичной системе счисления), нечетная (1 в двоичной системе счисления), то округление осуществляется в большую сторону.

Микропроцессор в зависимости от состояния флага RDM в регистре статуса ST2, выполняет математическое округление дробных чисел, если флаг RDM равен нулю, или банковское округление, если флаг RDM равен единице. Как математическое, так и банковское округление выполняется над числами, представленными в формате Q1.31 и Q9.31 с точностью, которая необходима для представления этих чисел в формат Q1.15.

Операция математического округления чисел в формате Q9.31 и Q1.31 до 16-разрядного числа в формате Q1.15 реализуется микропроцессором путем прибавления к округляемому числу константы 2^{-16}, или 00008000h (см. рисунок 1.9 и 1.11). В этом случае если последний отбрасываемый 15 разряд равен нулю, то число в формате Q1.15, размещенное в разрядах 16-31, не изменится. Если отбрасываемый разряд равен единице, то число в формате Q1.15 увеличится на величину $2^{-16} + 2^{-16}$, которая равна 2^{-15}, т.е. произойдет математическое округление.

Например, дробное число 00008001h, или 0,000015259254723787307739257812 5, при математическом округлении будет преобразовано в дробное число 00010001h, или 0,000030518043786287307739257812 5, а дробное число FFFE0001h, или -0,000061034690588712692260742187 5, – в дробное число FFFE8001h,

или -0,000076293479651212692260742187 5, что соответствует 16-разрядным дробным числам 0001h, или 0,000030517578125, и FFFEh, или -0,00006103515625.

Операция банковского округления чисел, представленных в форматах Q9.31 и Q1.31 до 16-разрядного дробного числа в формате Q1.15 реализуется микропроцессором в зависимости от значения младших 17 разрядов округляемого числа следующим образом:

– если младшие 16-разрядов округляемого числа находятся в диапазоне 0000h-7FFFh, то никаких действий над числом не производится (математическое округление);

– если младшие 16-разрядов округляемого числа равны 8000h, то при наличии единицы в предыдущем 17 разряде этого числа к нему прибавляется 0008000h, в противном случае число не изменяется (банковское округление);

– если младшие 16-разрядов округляемого числа находятся в диапазоне 8001h-FFFFh, то к этому числу также прибавляется 0008000h (математическое округление).

Например, дробное число 00018000h, или 0,0000457763671875, при банковском округлении будет округлено до 0002h, или 0,00006103515625, а дробное число 00008000h, или 0,0000152587890625, – до 0000h, или нуля.

1.1.4 Операционное устройство

Операционное устройство, или D модуль, – модуль микропроцессора, выполняющий основные операции по обработки сигналов и данных. В состав операционного устройства входят следующие блоки (рисунок 1.12):

– регистровый файл;
– арифметико-логический блок (арифметико-логическое устройство, АЛУ);
– сдвигатель;
– блок манипуляции битами (битовый блок);
– двойной умножитель-аккумулятор.

1.1.4.1 Регистровый файл

Регистровый файл операционного устройства предназначен для оперативного хранения обрабатываемых данных. Регистровый файл получает непосредственные данные, содержащиеся в коде команды, из буфера команд (I модуль), обменивается данными с устройством управления (I модуль) и устройством адресации (A модуль), а также получает данные из памяти по магистралям BB, CD и DB и передает данные в память по магистралям EB и FB. Данные, хранящиеся в регистровом файле, поступают

во все другие блоки операционного устройства, а также могут приниматься от этих устройств.

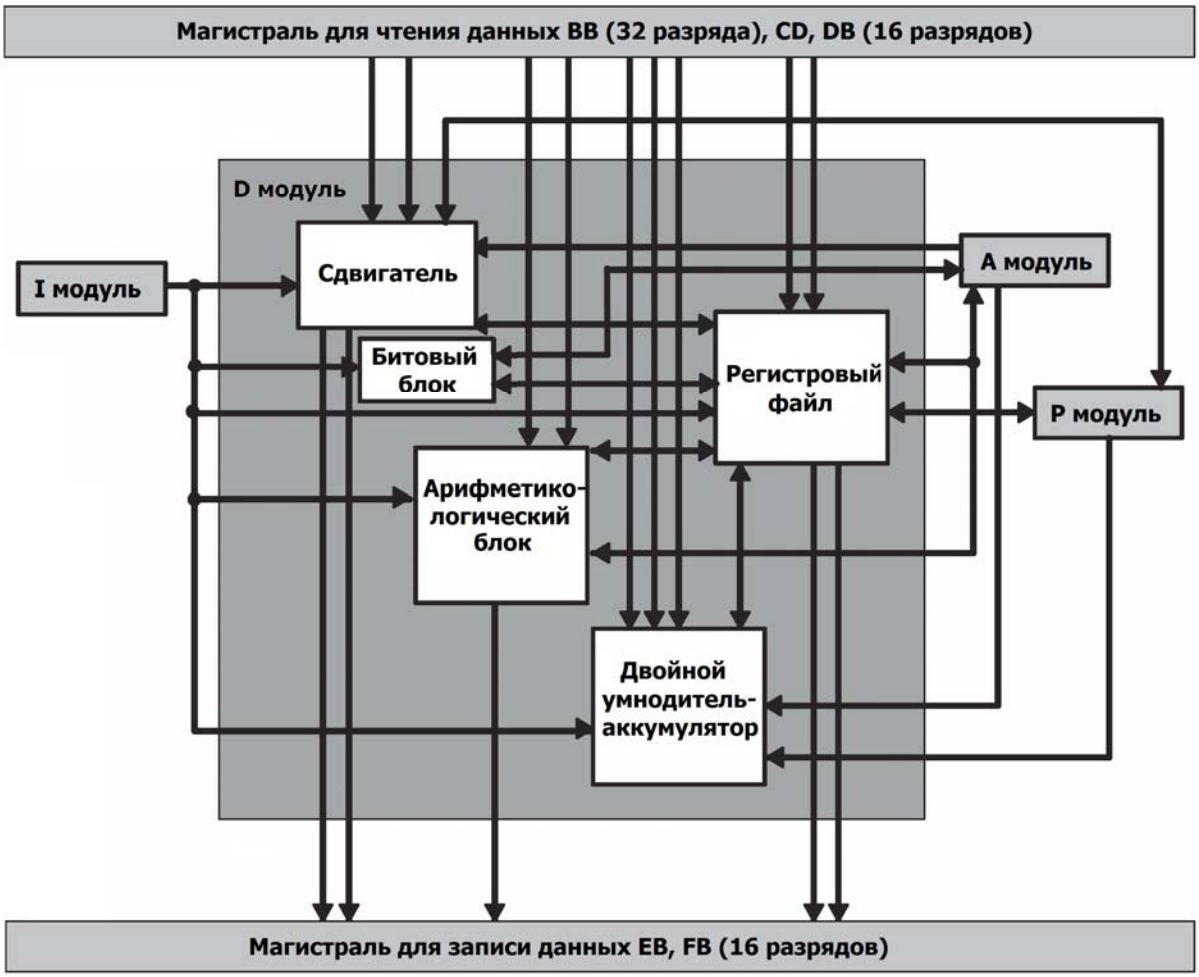


Рисунок 1.12 – Операционное устройство

В регистровом файле находятся четыре 40-разрядных регистра-аккумулятора AC0-AC3 и два 16-разрядных переходных регистра TRN0 и TRN1 (рисунок 1.13).

	39–32	31–16	15–0
AC0	AC0G	AC0H	AC0L
AC1	AC1G	AC1H	AC1L
AC2	AC2G	AC2H	AC2L
AC3	AC3G	AC3H	AC3L

	15–0
TRN0	
TRN1	

Рисунок 1.13 – Регистры операционного устройства

Аккумуляторы AC0-AC3 (англ. accumulator) являются составными регистрами и делятся на регистры младших 16 разрядов AC0L-AC3L (англ. low), регистры старших 16 разрядов AC0H-AC3H (англ. high) и регистры самых старших (сторожевых) 8 разрядов AC0G-AC3G (англ. guard) хранящегося в них 40-разрядного числа. Аккумуляторы используются для хранения операндов арифметико-логических команд, выполняемых арифметико-логическим устройством, команд умножения со сложением, выполняемых умножителями-аккумуляторами и команд сдвигов, выполняемых сдвигателем. Регистры AC0L-AC3L, AC0H-AC3H и AC0G-AC3G доступны по чтению и записи через адресное пространство памяти (см. приложение А).

Переходные регистры TRN0 и TRN1 (англ. transition) являются 16-разрядными и используются в командах MAXDIF и DMAXDIF (MINDIF и DMINDIF) – в командах сравнения и выбора максимального (минимального) значения в регистрах-аккумуляторах.

1.1.4.2 Арифметико-логический блок

Арифметико-логический блок принимает код операции из буфера команд (I модуль) и выполняет двустороннее взаимодействие с памятью через адресное пространство памяти, устройствами ввода-вывода через адресное пространство ввода-вывода, регистрами устройства адресации (A модуль), устройства управления (P модуль) и регистрами, находящимися непосредственно в регистровом файле операционного устройства. Кроме того, арифметико-логический блок может получать на вход результаты работы сдвигателя. Арифметико-логический блок операционного устройства может осуществлять одновременное выполнение двух 16-разрядных арифметических операций сложения и вычитания.

В зависимости от режима микропроцессора арифметико-логический блок обрабатывает 40-разрядные, 32-разрядные и 16-разрядные целые и дробные числа и может выполнять следующие арифметические, логические и битовые операции:

- инверсия разрядов – NOT (англ. not);
- поразрядная конъюнкция – AND (англ. and);
- поразрядная дизъюнкция – OR (англ. or);
- поразрядная неэквиваленция (исключающее ИЛИ) – XOR (англ. exclusion or);
- изменение знака – NEG (англ. negation);
- нахождение абсолютного значения – ABS (англ. absolute);
- сложение – ADD (англ. addition);
- вычитание – SUB (англ. subtraction);

– вычитание и сложение – SUBADD (англ. subtraction, addition);

– сложение и вычитание – ADDSUB (англ. addition, subtraction);

– сравнение – CMP (англ. comparison);

– сравнение – CMPAND (англ. comparison with AND);

– сравнение – CMPOR (англ. comparison with OR);

– округление – ROUND (англ. rounding);

– насыщение – SAT (англ. saturation);

– установка бит – BSET (англ. bit set);

– очистка бит – BCLR (англ. bit clear);

– инверсия бит – BNOT (англ. bit complement);

– проверка бит – BTST (англ. bit test);

– установка бит с проверкой BTSTSET (англ. bit test and set);

– очистка бит с проверкой BTSTCLR (англ. bit test and clear);

– выбор максимального и минимального значения – MAX, MAXDIF и MIN, MINDIF (англ. maximum, maximum difference, minimum, minimum difference).

1.1.4.3 Сдвигатель

Сдвигатель получает непосредственные данные команд из очереди команд (I модуль) и двусторонне взаимодействует с памятью, пространством ввода-вывода, регистрами устройства адресации (A модуль), устройства управления (P модуль) и операционного устройства (D модуль). Помимо этого сдвигатель выполняет сдвиги, которые необходимы для выполнения операций арифметико-логического блока операционного устройства (предварительная обработка операндов) и арифметико-логического блока устройства адресации (заключительная обработка результатов перед записью в регистровый файл).

Сдвигатель может выполнять:

– сдвиг 40-разрядных значений аккумуляторов: влево от одного до 31 разряда или вправо от одного до 32 разрядов, задаваемых в одном из временных регистров T0-T3 или в виде непосредственных данных в коде команды;

– сдвиг 16-разрядных значений регистров, ячеек памяти или регистров периферийных устройств, влево от одного до 15 разрядов, или вправо от одного до 16 разрядов, задаваемых в одном из временных регистров T0-T3 или в виде непосредственных данных в коде команды;

– сдвиг 16-разрядных непосредственных данных из кода команды влево от одного до 15 разрядов, задаваемых в виде непосредственных данных в коде этой же команды (генерация констант);

– циклический сдвиг значений в регистрах операционного устройства;

– округление и насыщение чисел в аккумуляторах перед их записью в память;

Сдвигатель обрабатывает 40-разрядные, 32-разрядные и 16-разрядные числа и может выполнять следующие операции (рисунки 1.14-1.16):

– логический сдвиг вправо или влево – SFTL (англ. shift logically);

– арифметический сдвиг вправо или влево – SFTS (англ. shift signed);

– условный сдвиг вправо или влево – SFTCC (англ. shift conditionally);

– циклический сдвиг влево – ROL (англ. rotate left);

– циклический сдвиг вправо – ROR (англ. rotate right).

Рисунок 1.14 – Логические сдвиги влево и вправо

Рисунок 1.15 – Арифметические сдвиги вправо и влево

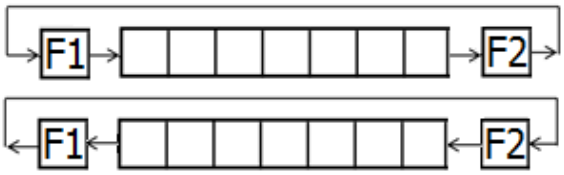


Рисунок 1.16 – Циклические сдвиги вправо и влево

(F1 и F2 – флаг переноса CARRY или флаг проверки и управления TC2)

При выполнении логических и арифметических сдвигов целое число сдвигов в формате со знаком задается в виде непосредственных данных в поле команды или во временных регистрах T0-T3. Циклические сдвиги всегда выполняются на один разряд.

1.1.4.4 Блок манипуляции битами

Для упрощения арифметико-логического блока и сдвигателя часть операций над числами и битовыми полями выполняет блок манипуляции битами:

- вычисление экспонент дробных чисел – EXP (англ. exponent);
- нормализация дробных чисел – MANT::NEXP (англ. mantissa and exponent);[1]
- извлечение битовых полей – BFXTR (англ. bit field extract);
- расширение битовых полей – BFXPA (англ. bit field expand);
- подсчет числа бит – BCNT (англ. bits count).

Экспонента числа – это целое число в диапазоне от –31 до 8, равное числу сдвигов, которые необходимо выполнить, чтобы преобразовать дробное число в формате Q9.31 в число в формате Q1.31 без потери значащих разрядов, а мантисса – это сами значащие разряды.

Экспонента дробного 40-разрядного числа равна номеру самого значащего бита (самой старшей единицы перед нулем – для отрицательных чисел, и самого старшего нуля перед единицей – для положительных чисел), считая со старшего разряда, минус восемь.[2]

Например, у 40-разрядного отрицательного дробного числа FFFFFFFFCBh самый значащий бит хранится в 33 разряде, считая от нуля слева. Следовательно, его экспонента равна –25. После сдвига на 25 разрядов влево имеем мантиссу CB000000h, представляющую число в формате Q1.31, которое необходимо умножить на 2^{-25}, чтобы получить исходное значение.

Извлечение и расширение битовых полей выполняются блоком манипуляции битами совместно со сдвигателем. Команды BFXTR и BFXPA имеют по два входных операнда: битовую маску, хранящуюся в коде команды, и битовое поле в одном из регистров-аккумуляторов, и один выходной операнда – регистр, куда помещается результат выполнения операции.

При извлечении битового поля битовая маска сканируется, начиная с самого младшего бита. Если в процессе сканирования в разряде N битовой маски найдена единица, то значение бита из разряда N битового поля заносится в текущий разряд результата, после чего текущим становится следующий слева разряд (рисунок 1.17, слева).

---

[1] Для указания на параллельное выполнение различными блоками микропроцессора двух частей одной команды (встроенный параллелизм) используется двойное двоеточие.

[2] Экспонента может быть подсчитана как число ведущих нулей – для положительных чисел, или число ведущий единиц – для отрицательных чисел, минус девять.

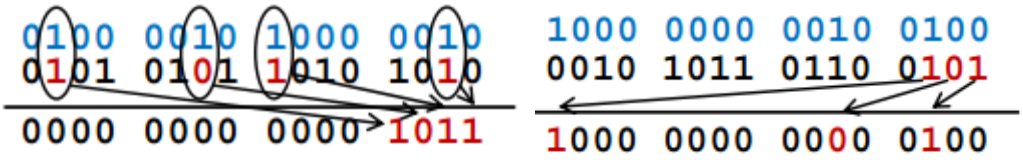


Рисунок 1.17 – Пример выполнения команд BFXTR (слева) и BFXPA (справа)

Расширение битовых полей выполняется обратным образом. Если в процессе сканирования битовой маски в разряде N найдена единица, то значение бита из текущего разряда битового поля заносится в разряд N результата, после чего текущим становится следующий слева разряд (рисунок 1.17, справа).

Команда BCNT определяет и записывает в один из временных регистров T0-T3 число разрядов, в которых встречается единица в результате конъюнкции двух каких-либо регистров-аккумуляторов AC0-AC3. При этом если число единиц четное, то один из флагов проверки и управления TC1-TC2 сбрасывается в ноль, если нечетное – то устанавливается в единицу.

1.1.4.5 Двойной умножитель-аккумулятор

Операционное устройство содержит два умножителя-аккумулятора (англ. multiple and accumulate). В каждом такте микропроцессора любой умножитель-аккумулятор осуществляет умножение двух 16-разрядных целых или дробных чисел, расширенных с учетом знака до 17 разрядов, и 40-битное сложение или вычитание результата умножения с текущим содержимым одного из аккумуляторов AC0-AC3.

Умножители-аккумуляторы могут получать непосредственные данные, хранящиеся в коде команды, из буфера команд (I модуль), значения из памяти или регистров периферийных устройств, а также из регистрового файла устройства адресации (A модуль). При выполнении операций умножения с накоплением умножители-аккумуляторы обмениваются данными с регистровыми файлами операционного устройства (D модуль) и устройства управления (P модуль).

В зависимости от режима микропроцессора умножитель-аккумулятор обрабатывает 16-разрядные целые или дробные числа, и выполняют следующие операции:

- умножение – MPY (англ. multiply);
- умножение со сложением – MAC (англ. multiply and accumulate);
- умножение с вычитанием – MAS (англ. multiply and subtract).

Входными операндами команд, выполняемых умножителями-аккумуляторами, могут быть (рисунок **1**.18):

– 17 разрядов регистров аккумуляторов AC0-AC3 (с 32 до 15);
– числа во временных регистрах T0-T3, расширенные до 17 разрядов;
– константы из кода команд, расширенные до 17 разрядов;
– содержимое 16-разрядной ячеек памяти, расширенное до 17 разрядов.

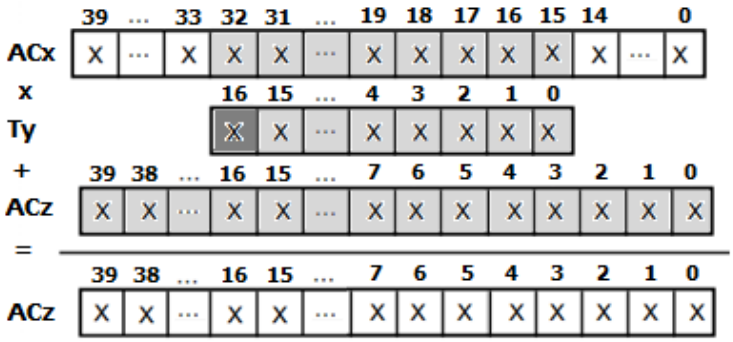


Рисунок **1**.18 – Умножение с накоплением

Результат выполнения операций умножения зависит от состояния флагов микропроцессора (см. пп. 1.1.7.6): флага дробного режима FRCT, флага 40-разрядного режима M40, флага округления RDM, флага насыщения операционного устройства SATD и флага насыщения при умножении SMUL. После выполнения операции устанавливается или сбрасывается один из флагов переполнения ACOV0-ACOV3.

Формат обрабатываемых данных определяется состоянием флага дробного режима FRCT: если флаг сброшен, то операнды интерпретируются как целые числа, если установлен, то как дробные. При умножении с накоплением контроль переполнения разрядной сетки осуществляется в 39 или 31 разряде аккумулятора в зависимости состояния флага M40. При установленном флаге SMUL (SATD) в процессе выполнения команд умножения с накоплением выполняется насыщение результата умножения (накопления). В зависимости от состояния флага округления RDM, производится математическое или банковское округление результата умножения.

Помимо одиночных команд двойной умножитель-аккумулятор параллельно выполняет:

1) два умножения MPY, умножение со сложением MAC или умножение с вычитанием MAS: MAC::MAC, MAC::MAS, MAC::MPY, MAS::MAC, MAS::MAS, MAS::MPY, MPY::MAC, MPY::MAS, MPY::MPY;

2) совместно с устройством адресации умножение MPY, умножение со сложением MAC или умножение с вычитанием MAS, и пересылку данных MOV (англ.

move) или модификацию адресного регистра AMAR (англ. address modify in auxiliary register): MACM::MOV, MASM::MOV, MPYM::MOV, AMAR::MAC, AMAR::MAS, AMAR::MPY,[3]

3) совместно с арифметико-логическим блоком операционного устройства комплексные команды для симметричной FIRSADD (англ. finite impulse response addition) и кососимметричной фильтрации FIRSSUB (англ. finite impulse response subtraction), а также комплексную команду для адаптивной фильтрации сигналов LMS (англ. least mean square).[4]

Для обеспечения параллельного выполнения двух операций в одном такте микропроцессора входные данные в умножители-аккумуляторы поступают одновременно по магистралям DB и CB, а также по различным частям магистрали BB (рисунок **1**.19).

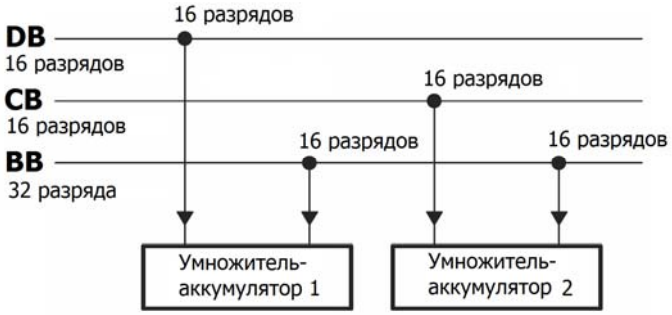


Рисунок **1**.19 – Входы умножителей-аккумуляторов

1.1.5 Устройство адресации

Устройство адресации (A модуль) – модуль микропроцессора, отвечающий за генерацию адресов данных в пространстве адресов памяти и ввода-вывода. В отличие от операционного устройства (D модуль) устройство адресации не обрабатывает данные и не управляет их пересылкой по шинам микропроцессора, а лишь генерирует адреса операндов выполняемых команд. У микропроцессора нет никакого другого устройства, которое управляет адресными шинами для чтения и записи данных.

Устройство адресации выполняет следующие функции:

- хранит адреса и их компоненты в своем регистровом файле;
- выполняет арифметические, логические и сдвиговые операции над адресами;
- реализует различные способы адресации операндов в командах;

[3] Адресные регистры принадлежат регистровому файлу устройства адресации и описаны в пп. 1.1.5.1.
[4] Описание перечисленных команд фильтрации сигналов дано в п. 2.2.2 и п. 2.2.4.

– формирует адреса в пространстве адресов памяти и ввода-вывода.

Устройство адресации состоит из регистрового файла, арифметико-логического блока и генератора адресов данных (рисунок 1.20).

1.1.5.1 Регистровый файл

Регистровый файл устройства адресации получает хранящиеся в нем данные из буфера команд (I модуль), обменивается данными с регистровыми файлами устройства управления (P модуль) и операционного устройств (D модуль), получает и передает данные на магистрали данных при обмене с памятью и регистрами периферийных устройств. Внутри устройства адресации регистровый файл имеет двухстороннее взаимодействие с генератором адресов данных и арифметико-логическим устройством.

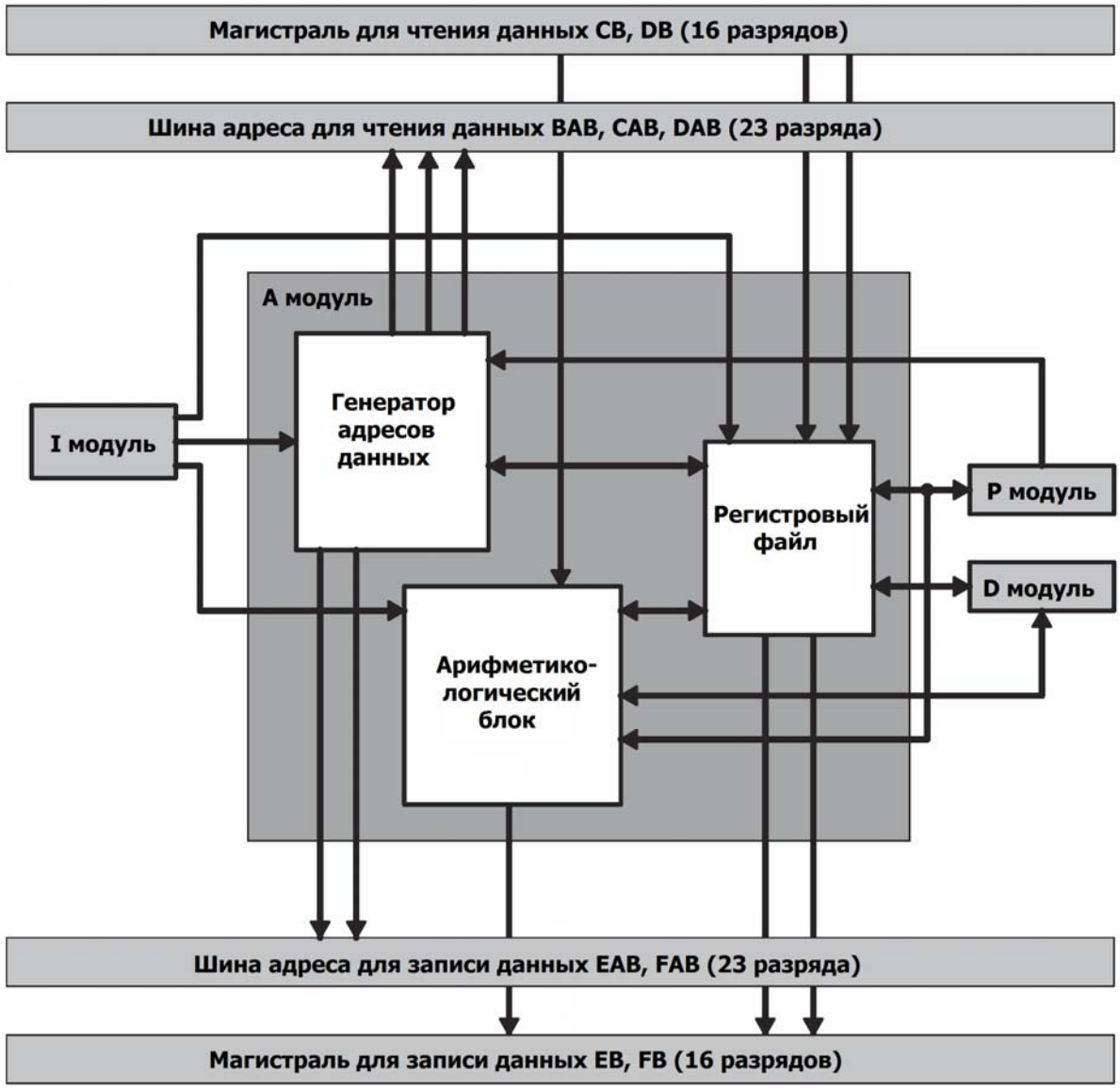


Рисунок 1.20 – Устройство адресации

Регистровый файл включает в себя следующие группы регистров (рисунок 1.21):

1) регистры страниц:

– XDP (англ. extended data page) – 23-разрядный расширенный регистр страницы данных, состоящий из младшего 16-разрядного регистра DP и старшего 7-разрядного регистра DPH (англ. high);

– PDP (англ. pperipheral data page) – 9-разрядный регистр страницы данных периферийных устройств;

2) регистры-указатели:

– XAR0-XAR7 (англ. extended auxiliary register) – 23-разрядные расширенные дополнительные регистры-указатели, состоящие из младших 16-разрядных регистров AR0-AR7 и старших 7-разрядных регистров страниц AR0H- AR7H;

– XCDP (англ. extended coefficient data pointer) – 23-разрядный расширенный регистр-указатель коэффициентов, состоящий из младшего 16-разрядного регистра CDP и старшего 7-разрядного регистра страницы CDPH;

– XSP (англ. extended stack pointer) – 23-разрядный расширенный регистр-указатель стека данных, состоящий из младшего 16-разрядного регистра SP и старшего 7-разрядного регистра страницы SPH;

– XSSP (англ. extended system stack pointer) – 23-разрядный расширенный регистр-указатель системного стека, состоящий из младшего 16-разрядного регистра SSP и старшего 7-разрядного регистра страницы SPH;[5]

3) регистры циклического буфера:

– BK03, BK47, BKC (англ. buffer size) – 16-разрядные регистры размера буфера, применяемые совместно с регистрами-указателями XAR0-XAR3, XAR4-XAR7, XCDP соответственно;

– BSA01, BSA23, BSA45, BSA67, BSAC (англ. buffer start address) – 16-разрядные регистры адреса (смещения) начала буфера относительно регистров-указателей XAR0-XAR1, XAR2-XAR3, XAR4-XAR5, XAR6-XAR7, XCDP соответственно;

4) временные регистры:

– T0-T3 (англ. temporary) – 16-разрядные временные регистры.

[5] В состав регистров XSP и XSSP входит один и тот же 7-разрядный регистр старшей части адреса SPH.

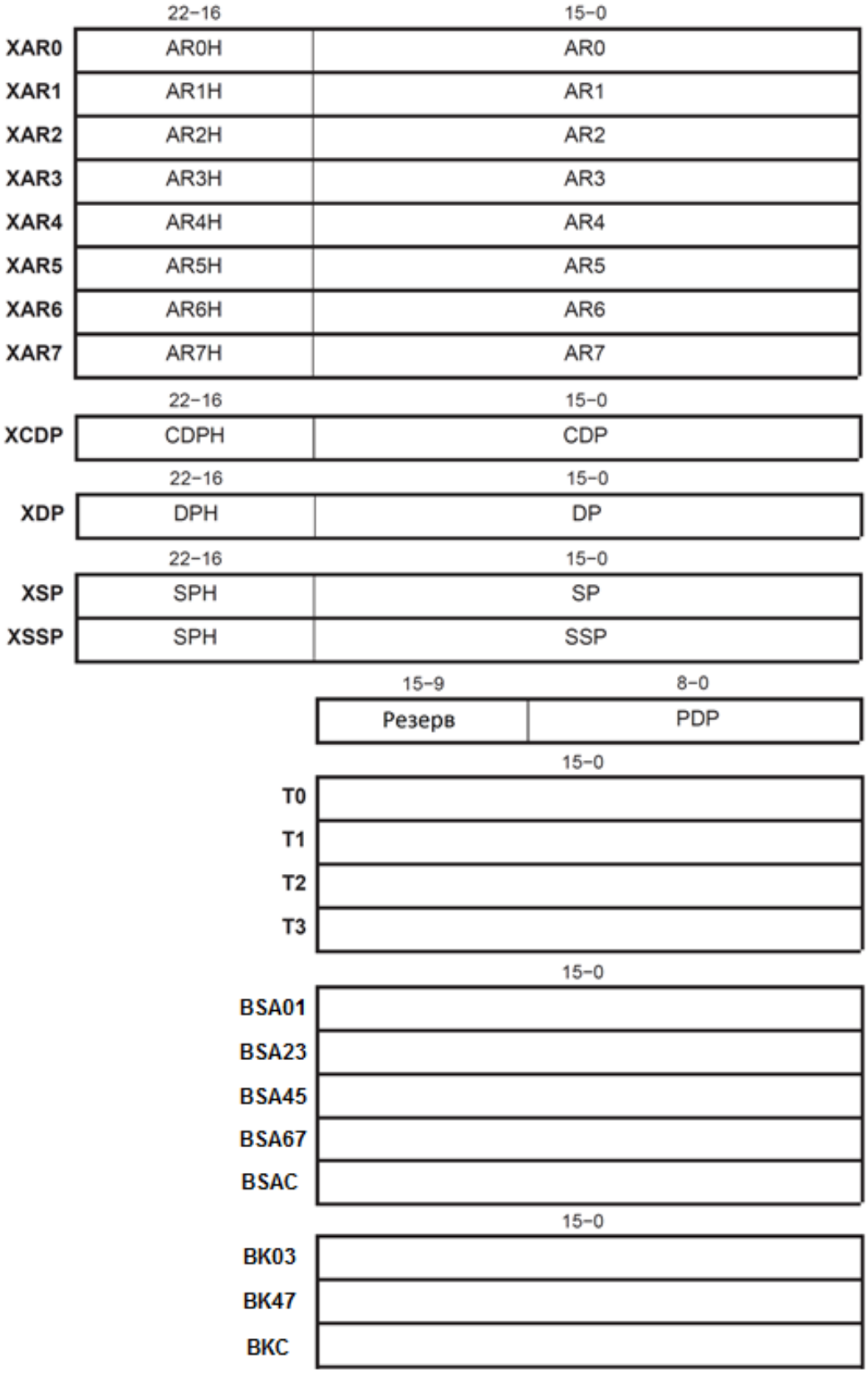


Рисунок 1.21 – Регистры устройства адресации

Регистр страницы данных XDP состоят из двух частей: старшей части DPH, в которой хранится номер текущей страницы данных в памяти, и младшей части DP, где

задается смещение текущего слова внутри этой страницы. Регистр XDP используется в прямом методе адресации данных, при котором 23-разрядный адрес слова определяется как содержимое регистра XDP, сложенное со 7-разрядным смещением, задаваемым непосредственно в коде команды. Регистр DPH используется в абсолютном методе адресации, при котором 23-разрядный адрес слова состоит из старшей 7-разрядной части, расположенной в регистре DPH, и младшей 16-разрядной части, заданной непосредственно в коде команды. Регистр XDP доступен по чтению и записи только с помощью команды пересылки адресных регистров AMOV (англ. address move), а регистры DPH и DP – с помощью команд пересылки данных MOV (англ. move) и через адресное пространство памяти по адресам 00002Bh и 00002Eh (см. приложение А).

Регистр страницы данных периферийных устройств PDP используется в прямом методе адресации регистров периферийных устройств. В этом случае номер текущей 128-словной страницы хранится в регистре PDP, а 7-разрядное смещение слова внутри этой страницы задается непосредственно в коде команды. Регистр PDP доступен по чтению и записи только через адресное пространство памяти по адресу 00002Fh (см. приложение А).

Регистры-указатели стеков XSSP и XSP содержат адреса слов на вершинах системного стека и стека данных, и состоят из двух частей – общего для двух стеков 7-разрядного регистра SPH с адресом страницы, и младших 16-разрядных регистров SSP и SP со смещением вершин стека в этой странице. Регистр XSSP используется при вызове подпрограмм для сохранения старшей части адреса возврата и состояния микропроцессора, а регистр XSP – для сохранения младшей части адреса возврата, записи аргументов подпрограмм в стек и доступа к этим аргументам из подпрограмм (см. пп. 1.1.7.3). Регистр XSP, так же как и регистр XDP, используется в прямом методе адресации данных, при котором 23-разрядный адрес слова определяется как содержимое регистра XSP, сложенное со 7-разрядным смещением, задаваемым непосредственно в коде команды. Выбор регистра XDP или XSP при вычислении адреса операнда при его прямой адресации задается флагом CPL в регистре статуса ST1 (см. пп. 1.1.7.6).

Регистры XSSP и XSP доступны по чтению и записи только с помощью команды пересылки адресных регистров AMOV, регистр DPH – только через адресное пространство памяти, а регистры SSP и SP – с помощью команд пересылки данных MOV и через адресное пространство памяти (см. приложение А).

Дополнительные регистры-указатели XAR0-XAR7 и регистр-указатель коэффициентов XCDP состоят из двух частей: старшей части AR0H-AR7H и CDPH, в которой хранится номер страницы в памяти, и младшей части AR0-AR7 и CDP, где задается смещение слова внутри страницы. Регистры AR0-AR7 и CDP могут использоваться как регистры общего назначения для хранения 16-разрядных данных, а также для хранения адресов бит. Однако основное их назначение – участие в формировании адресов операндов в косвенных способах адресации данных (см. пп. 1.1.6). Регистры XAR0-XAR7 и XCDP доступны по чтению и записи только с помощью команды пересылки адресных регистров AMOV, регистры AR0H-AR7H и CDPH – только через адресное пространство памяти, а регистры AR0-AR7 и CDP – с помощью команд пересылки данных MOV и через адресное пространство памяти (см. приложение А).

Регистры циклического буфера позволяют организовать циклический (кольцевой) доступ к пяти различным буферам, младшая 16-разрядная часть адреса которых загружается в регистры BSA01, BSA23, BSA45, BSA67 и BSAC, а старшая 7-разрядная часть – берется из регистров страниц AR0H или AR1H, AR2H или AR31H, AR4H или AR5H, AR6H или AR7H, и CDPH соответственно (рисунок 1.22).

Длина буферов в словах загружается в регистры BK03, BK47 и BKC, причем в регистре BK03 хранится длина буфера с начальным адресом в BSA01 и BSA23, в регистре BK47 – длина буфера с начальным адресом в BSA45 и BSA67, а в BKC – длина буфера с начальным адресом в BSAC.

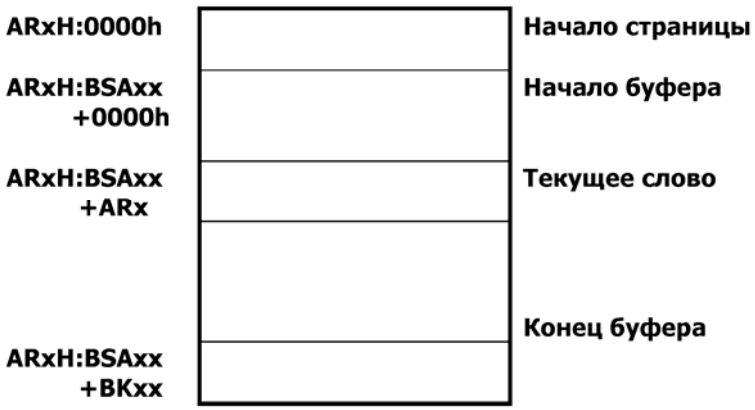


Рисунок 1.22 – Циклический буфер

При организации циклической адресации внутри выделенных буферов используются дополнительные регистры AR0-AR7 и CDP, которые в этом случае содержат положительное смещение текущего слова относительно начала буфера,

адресуемого парами регистров AR0H:BSA01, AR1H:BSA01, AR2H:BSA23, AR3H:BSA23, AR4H:BSA45, AR5H:BSA45, AR6H:BSA67, AR7H:BSA67 и CDPH:BSAC соответственно.

При достижении смещения в регистрах AR0-AR7 и CDP значений, содержащихся в регистрах BK03 (для AR0-AR3), BK47 (для AR4-AR7), BKC (для CDP), соответствующий дополнительный регистр-указатель или регистр-указатель коэффициентов обнуляется, т.е. происходит возврат в начало циклического буфера.

Временные регистры T0-T3 предназначены для хранения следующих данных:

– одного из операндов команд умножения и умножения с накоплением;

– числа дополнительных сдвигов, выполняемых операционных устройством и используемых в командах сложения, вычитания и загрузки регистров;

– дополнительных данных, необходимых для выполнения двух параллельных 16-разрядных операций операционного устройства.

1.1.5.2 Арифметико-логический блок

Арифметико-логический блок является 16-разрядным и используется для выполнения операций над адресами и их компонентами:

– арифметических операций сложения, вычитания и сравнения;

– логических поразрядных операций конъюнкции, дизъюнкции и отрицания;

– операций циклического, логического и арифметического сдвига;

– битовых операций проверки, установки, сброса и инверсии;

– операций по модификации регистров в косвенных методах адресации.

Арифметико-логический блок может получать непосредственные данные из буфера команд (I модуль), обмениваться данными с регистровыми файлами устройства адресации (A модуль) и операционного устройства (D модуль), получать и передавать данные на магистрали данных для обмена с памятью и регистрами периферийных устройств.

1.1.5.3 Генератор адресов данных

Генератор адресов данных ответственен за вычисление и передачу текущих адресов на шины адреса для чтения и записи данных, для чего он может получать непосредственные данные из буфера команд (I модуль) и содержимое регистров регистрового файла устройства адресации (A модуль), а от устройства управления (P модуль) – состояния флагов циклической адресации AR0LC-AR7LC и CDPLC.

1.1.6 Методы адресации данных

Микропроцессор поддерживает 8 методов адресации операндов, представленных на рисунке 1.23, где знаком + обозначена операция арифметического сложения, а знаком ⊕ – сложение по модулю, равному размеру циклического буфера.

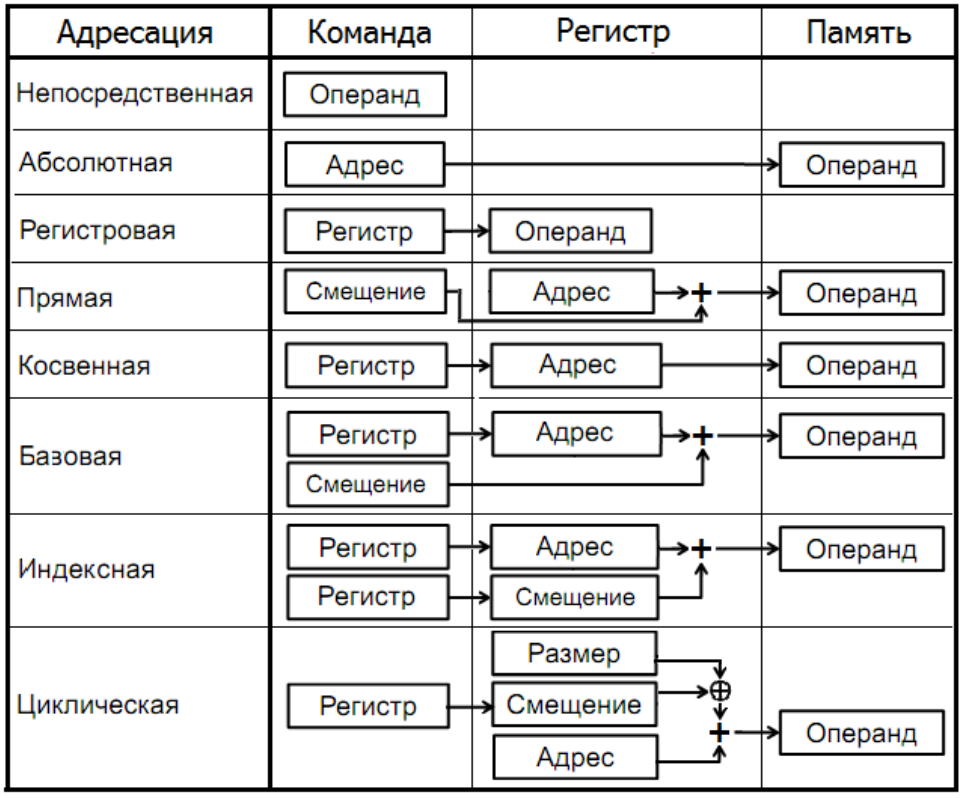


Рисунок 1.23 – Методы адресации операндов

Для задания метода адресации в коде команды выделяются поля (группы смежных разрядов), куда записываются данные, необходимые для формирования адресов операндов. Например, команда с мнемоническим обозначением

MOV XAsrc, dbl(Lmem),

предназначенная для пересылки содержимое 32-разрядного регистра XAsrc в двойное слово памяти по адресу Lmem, имеет трехбайтовый формат

11101101 AAAAAAAI XSSS0101,

где двоичные цифры 0 и 1 задают в соответствующих разрядах код команды, а буквы – разряды команды, куда могут быть записаны данные, ее модифицирующие:

– поле I – флаг метода адресации операнда в памяти (0 – прямая адресация, 1 – косвенная, базовая или индексная адресация);

– поле AAAAAAA – в зависимости от значения поля I или 7-разрядное смещение для прямого метода адресации, или код способа косвенной, базовой или индексной адресации;

– поле XSSS – 4-разрядный номер регистра (с 0000 по 0011 – регистры-аккумуляторы AC0-AC3, 0100 – регистр XSP, 0101 – регистр XSSP, 0110 – регистр XDP, 0111 – регистр XCDP, с 1000 по 1111 – регистры XAR0-XAR7).

Микропроцессор поддерживает два типа доступа к операндам – доступ к адресуемым ячейкам памяти, или адресацию памяти, и доступ к адресуемым разрядам в регистрах микропроцессора, или адресацию бит.

При адресации памяти формируется 23-разрядный адрес операнда в адресном пространстве памяти микропроцессора или 16-разрядный адрес регистра периферийного устройства в адресном пространстве ввода-вывода. При адресации бит в качестве адреса бита используется 7-разрядное число, равное номеру разряда в регистре микропроцессора, например: AC0, AR5, T0, или в регистровой паре, например: AC2-AC3, T0-T1, AR2-AR3.

1.1.6.1 Непосредственная и регистровая адресация

При непосредственной адресации операнд содержится в полях команд:

– в поле k23 – 23-разрядная адресная константа;

– в поле K16 (k16) – 16-разрядное число со знаком (без знака);

– в поле k12 – 12-разрядное число без знака;

– в поле k9 – 9-разрядное число без знака;

– в поле K8 (k8) – 8-разрядное число со знаком (без знака);

– в поле k5 – 5-разрядное число без знака;

– в поле k4 – 4-разрядное число без знака;

– в поле k3 – 3-разрядное число без знака.

При регистровой адресации в полях команды кодируется номер регистра, содержащего операнд:

– в полях DD и SS – 2-разрядный номер регистра аккумулятора AC0-AC3;

– в полях dd и ss – 2-разрядный номер временного регистра T0-T3;

– в полях XXX и YYY – 3-разрядный номер дополнительного регистра AR0-AR7;.

– в полях FDDD и FSSS – 4-разрядный номер регистра из списка AC0, AC1, …, AC3, T0, T1, …, T3, AR0, AR1, …, AR7;

– в полях XDDD и XSSS – 4-разрядный номер регистра из списка AC0, AC1, …, AC3, XSP, XSSP, XDP, XCDP, XAR0, XAR1, …, XAR7.

Примеры непосредственной и регистровой адресации:

– MOV #1F, DPH – загрузка в регистр страницы данных DPH значения 31;

– MACK T0, #-123, AC0 – умножить содержимое регистра T0 на константу -123 и прибавить полученный результат к содержимому аккумулятора AC0;

– AMOV #123456h, XSP – загрузка в регистр XSP адреса 123456h;

– BSET #10, ST3 – установка в единицу 10 разряда регистра статуса ST3;.

– BCLR AR0, AC0 – сброс в ноль разряда регистра-аккумулятора AC0, номер которого находится в 6 младших разрядах дополнительного регистра AR0.

1.1.6.2 Абсолютная адресация

При абсолютной адресации памяти используется дополнительное поле команды a23 (a16), которое имеет длину 3 (2) байта и непосредственно следует за командой. В поле a23 содержится полный адрес операнда в памяти, а в поле a16 – только его младшая часть. Старшая часть адреса в последнем случае извлекается из регистра страницы данных DPH. Если в команде производится обращение в адресное пространство ввода-вывода, то поле a16 содержит полный адрес регистра периферийного устройства. Абсолютная адресация бит микропроцессором не поддерживается.

Для задания абсолютного 23-разрядного адреса памяти и адреса бита используется адресная константа, перед которой ставятся знаки * и #, а для задания 16-разрядного адреса – ключевое слово abs16, после которого в круглых скобках приводится адресная константа, например: *#123456h, abs16(#1234h).

Примеры абсолютной адресации:

– MOV dbl(*#123456h), pair(T0) – загрузка в регистровую пару T0:T1 двойного слова, находящегося в памяти по адресу 123456h;

– PSH *abs16(#1234h) – занести на вершину стека слово, расположенное в памяти по адресу DPH:1234h;

– OR #0010h, port(#3210h) – установить в единицу 4 разряд регистра периферийного устройства с адресом 3210h.

1.1.6.3 Прямая адресация

При прямой адресации памяти в 7-разрядном поле команды AAAAAAA содержится смещение операнда относительно адреса, размещенного в регистрах-указателях XDP, XSP или PDP. Регистр XDP используется как базовый, когда флаг CPL

в регистре статуса микропроцессора ST1 сброшен, а регистр XSP – при установленном флаге CPL (см. пп. 1.1.7.6). В свою очередь прямая адресация регистров периферийных устройств осуществляется относительно регистра PDP и не зависит от состояния флага CPL. При прямой битовой адресации регистры-указатели не используются, в поле команды AAAAAAA задается первый операнд – 7-разрядный номер разряда регистра, сам же регистр кодируется в другом поле как второй операнд команды.

Для указания смещения операнда при прямой адресации используется знак @ и число, например: @31, @#0FEDBh.

Примеры прямой адресации:

– MOV @#–12, T0 – загрузка в регистр T0 слова, расположенного в памяти по адресу XDP–12 или XSP–12 в зависимости от состояния флага CPL;

– MOV low_byte(@#12h) << #-3, AC1 – загрузка в аккумулятор AC1 младшего байта слова, расположенного в памяти по адресу XDP+12h или XSP+12h в зависимости от состояния флага CPL, и сдвинутого логически вправо на 3 двоичных разряда;

– AND #FFEFh, port(@#32h) – сбросить в ноль 4 разряд регистра периферийного устройства с адресом PDP:00h+32h;

– BTSTP @30, AC3 – проверка битовой пары в регистре-аккумуляторе AC3 с номерами разрядов 30 и 31 и копирование их значений во флаги проверки и управления TC1 и TC2;

– BNOT @#0Ch, T3 – инвертирование 12 разряда 16-разрядного регистра T3.

1.1.6.4 Косвенная адресация

При косвенной адресации адрес операнда содержится в регистре, номер которого кодируется в одном из полей команды. Для косвенной адресации используются 16-разрядные регистры-указатели AR0-AR7 и CDP. Полный 23-разрядный адрес операнда в памяти формируется с учетом соответствующего регистра страницы AR0H-AR7H и CDPH. Для косвенной адресации бит регистры страниц не используются: номер разряда хранится в младших разрядах регистров-указателей AR0-AR7 и CDP.

Различают обычный и модифицирующие способы косвенной адресации, а именно: автоинкрементный или автодекрементный способ косвенной адресации, при котором после или до чтения-записи операнда содержимое 16-разрядного регистра-указателя увеличивается или уменьшается на длину адресуемой ячейки памяти. Если модификация регистра-указателя осуществляется до операции чтения-записи операнда, то такая адресация называется префиксной, если после – постфиксной.

Признаком косвенной адресации является наличие звездочки перед именем регистра, например, *AR0. Для указания на автоинкрементную префиксную (постфиксную) косвенную адресацию перед (после) имени регистра записывается знак плюс, а на автодекрементную – знак минус, например: *CDP+, *CDP–, *+AR5, *–AR5.[6]

Примеры косвенной адресации:

– MOV *CDP, T2 – загрузка в 16-разрядный регистр T2 содержимого слова памяти с адресом в регистре XCDP;

– POP *AR0+ – извлечение из стека слова и его запись в ячейку памяти с адресом в регистре XAR0 и увеличение содержимого регистра XAR0 на единицу;

– SUB dual(*AR1–), AC1 – вычитание из содержимого аккумулятора двойного слова с адресов в регистре XAR1 и уменьшение содержимого регистра XAR1 на два;

– AMAR *+AR3 – модификация регистра путем выполнения закодированного в команде метода адресации, или увеличение содержимого XAR3 на единицу;

– MPY *–AR6, *CDP+, AC1 – умножение операнда-слова по адресу AR6H:(AR6–1) на операнд-слово по адресу в регистре XCDP, сохранение результата умножения в аккумуляторе AC1, уменьшение (увеличение) содержимого регистра-указателя XAR6 (XCDP) на единицу;

– BSET *AR2, AC1 – установка в единицу разряда регистра-аккумулятора AC1, номер которого находится в регистре-указателе XAR2;

– BCLR *AR2+, AC1 – сброс в ноль разряда регистра-аккумулятора AC1, номер которого находится в регистре-указателе XAR2, и увеличение содержимого регистра XAR2 на единицу;

– BNOT *–AR2, AC1 – инвертирование разряда регистра-аккумулятора AC1 с номером, равным содержимому регистра-указателя XAR2 минус единица, и уменьшение содержимого регистра XAR2 на единицу.

1.1.6.5 Базовая адресация

При базовой адресации в коде команды задается базовый регистр-указатель и смещение операнда. В качестве базовых регистров-указателей используются регистры-указатели AR0-AR7 и CDP, а смещение задается в поле K16, которое непосредственно следует за командой. Адрес операнда вычисляется как сумма содержимого базового 16-разрядного регистра-указателя и 16-разрядного смещения со знаком, а полный 23-

[6] Префиксная автоинкрементная и префиксная автодекрементная косвенная адресация через регистр-указатель CDP микропроцессором не поддерживается.

разрядный адрес формируется с учетом соответствующего регистра страницы AR0H-AR7H и CDPH.

Различают обычную и модифицирующую базовую адресацию. При обычной базовой адресации содержимое регистра-указателя не изменяется. При модифицирующей базовой адресации вычисленный адрес операнда становится новым содержимым используемого базового регистра-указателя.

Признаком базовой адресации является наличие звездочки перед именем регистра и знаковой константы в круглых скобках после имени регистра, например, *CDP(#32). Для указания на модифицирующую базовую адресацию перед именем регистра указывают два знака: звездочку и плюс, например, *+AR3(#56).

Примеры базовой адресации:

– MOV AR3, high_byte(*CDP(#4)) – запись старшего байта регистра AR3 в слово памяти с адресом CDPH:(CDP+4);

– CMP *+AR4(#27) == #-237, TC2 – сравнение содержимого слова в памяти по адресу AR4H:(AR4+27) с непосредственной константой –237, сохранение результата сравнения во флаге проверки и управления TC2 и увеличение содержимого регистра XAR4 на 27;

– BCLR *AR4(#31), AC2 – сброс с ноль разряда регистра-аккумулятора AC2 с номером, равным XAR4+31;

– BSET *+AR4(#3), AC2 – установка в единицу разряда регистра-аккумулятора AC2 с номером, равным XAR4+3, и увеличение содержимого регистра-указателя XAR4 на 3.

1.1.6.6 Индексная адресация

При индексной адресации адрес операнда вычисляется как сумма значений двух регистров: базового регистра-указателя XAR0-XAR7 и индексного регистра T0-T1. Перед сложением с базовым регистром индексный регистр расширяется с учетом знака до 23-разрядного числа.

Различают обычную, модифицирующую и бит-реверсивную индексную адресацию. При обычной индексной адресации содержимое базового регистра не изменяется. При модифицирующей индексной адресации вычисленный адрес операнда становится новым содержимым используемого базового регистра. Для ускорения быстрого преобразования Фурье используется бит-реверсивная индексная адресация, задаваемая одним из регистров-указателей XAR0-XAR7 и индексным регистром T0 [36, с. 203]. В этом случае после чтения-записи операнда новый адрес, заносимый в

базовый регистр, вычисляется в бит-реверсивной манере, при которой веса разрядов регистра T0 изменяются: вес нулевого разряда становится равным -2^{15}, а вес 15 разряда – 2^0 (рисунок 1.24).

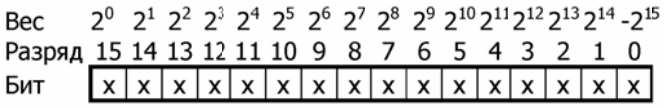


Рисунок 1.24 – Бит-реверсивный формат числа со знаком

Признаком обычной индексной адресации является указание знака * перед именем регистра-указателя AR0-AR7 и задание имени индексного регистра в круглых скобках после имени базового регистра, например, *AR3(T1). Для указания на модифицирующую индексную адресацию имена базового и индексного регистра, соединенные знаком + или –, помещаются в круглые скобки, перед которыми ставится знак *, например: *(AR3+T0), *(AR7–T1). Для указания на бит-реверсивную индексную адресацию после имени индексного регистра указывается буква B, например: *(AR3+T0B), *(AR7–T0B).

Примеры индексной адресации:

– MOV *AR1(T1) << #16, AC0 – загрузить регистр-аккумулятор AC0 слово по адресу XAR1+T1, сдвинутое на 16 разрядов влево;

– MAS *(AR6–T0), *CDP, AC2 – умножить два слова с адресами XAR6–T0 и XCDP, вычесть результат умножения от содержимого регистра-аккумулятора AC2 и сохранить адрес первого множимого XAR6–T0 в регистре XAR6;

– MOV HI(AC2), *(AR6+T0) – записать в ячейку памяти по адресу XAR6+T0 старшего слова регистра-аккумулятора AC2 и сохранить адрес этой ячейки памяти в регистре XAR6;

– MOV AC0, dbl(*(AR1+T0B)) – записать двойное слово из регистра-аккумулятора AC0 в две смежные ячейки памяти с начальным адресом XAR1+T0, после чего загрузить в регистр XAR1 новое значение адреса XAR1+T0B, вычисленное в бит-реверсивной манере;

– BSET *(AR2–T0), AC3 – установка в единицу разряда регистра-аккумулятора с номером, равным XAR2–T0, и загрузка этого номера в регистр-указатель XAR2;

– BCLR *AR2(T0), AC3 – сброс в ноль разряда регистра-аккумулятора с номером, равным XAR2+T0.

– BNOT *(AR2+T0B), AC3 – инвертирование разряда регистра-аккумулятора с номером, равным XAR2+T0, и загрузка в регистр-указатель XAR2 нового номера, XAR2+T0B, вычисленного в бит-реверсивной манере.

1.1.6.7 Циклическая адресация

При циклической адресации любой из дополнительных регистров-указателей AR0-AR7 и регистр-указатель коэффициентов CDP независимо друг от друга могут быть переведены в линейный или циклический режим адресации. В линейном режиме вычисление адреса операнда и его модификация осуществляется обычным образом, а в циклическом режиме – с учетом начального адреса буфера и его длины (рисунок 1.25).

Рисунок 1.25 – Циклическая адресация

Для задания размера циклических буферов используются регистры BK03, BK47 и BKC. Размер буфера измеряется в словах – для операций чтения-записи слов, или в битах – для битовых операций. Начальный адрес буферов хранится в регистрах BSA01, BSA23, BSA45, BSA67 и BSAC. Взаимное соответствие регистров-указателей, регистров страниц, регистров начального адреса буфера и регистров длины буфера приведено в таблице 1.3.

Из таблицы видно, что одновременно можно организовать до 5 циклических буферов с тремя различными размерами.[7] Циклический буфер, образованный регистрами AR0-AR7, может иметь до двух различных указателей чтения-записи, а

[7] На самом деле при загрузке в регистры AR0H-AR7H и CDPH различных номеров страниц можно организовать девять циклических буферов. В этом случае размер буферов может принимать одно из трех значений, а начальный адрес (смещение) внутри страниц – одно из пяти возможных значений.

циклический буфер, образованный регистром CDP, – только один.[8] Включение режима циклической адресации относительно регистров AR0-AR7 и CDP осуществляется установкой флагов AR0LC-AR7LC и CDPLC в регистре статуса микропроцессора ST2 (см. пп. 1.1.7.6).

Таблица 1.3 – Организация циклических буферов

Указатель	Страница	Начальный адрес	Размер
AR0	AR0H	BSA01	BK03
AR1	AR1H	BSA01	BK03
AR2	AR2H	BSA23	BK03
AR3	AR3H	BSA23	BK03
AR4	AR4H	BSA45	BK47
AR5	AR5H	BSA45	BK47
AR6	AR6H	BSA67	BK47
AR7	AR7H	BSA67	BK47
CDP	CDPH	BSAC	BKC

Совместно с командами обращения к циклически буферам можно использовать квалификаторы циклической .CR или линейно адресации .LR.[9] В этом случае текущие состояния флагов AR0LC-AR7LC и CDPLC не учитываются, например:

– ADD.CR dual(*CDP+), AC0, AC1 – сложение двойного слова по адресу CDPH:(BSAC+CDP) с содержимым регистра-аккумулятора AC0, сохранение результата сложения в регистре-аккумуляторе AC1 и циклическая модификация регистра-указателя CDP после его увеличения на два;

– SUB.LR uns(*–AR0), AC0, AC1 – вычитание из содержимого регистра-аккумулятора AC0 значения слова по адресу AR0H:(AR0–1), интерпретируемого в формате без знака, сохранение результата вычитания в регистре-аккумуляторе AC1 и уменьшение содержимого регистра-указателя AR0 на один.

При организации битовых циклических буферов в регистрах начального адреса BSA01, BSA23, BSA45, BSA67 и BSAC задается номер начального бита, а в регистрах размера буфера BK03, BK47 и BKC – длина буфера в битах. В этом случае в регистрах-указателях AR0-AR7 и CDP хранится относительная позиция текущего бита, а

[8] Возможно совмещение нескольких циклических буферов, адресуемых различными регистрами-указателями. В предельном случае можно организовать один циклический буфер с девятью указателями на текущее слово.

[9] Квалификаторы линейной и циклической адресации представляют собой однобайтовые префиксы, размещаемые в потоке команд непосредственно перед командой: .CR – 9Dh, а .LR – 9Ch.

регистры страниц AR0-AR7 и CDP не используются, так как битовый буфер всегда размещается в регистрах микропроцессора.

Примеры циклической адресации:

– SQR *+AR0(#12), AC3 – возвести в квадрат слово, размещенное по адресу AR0H:(BSA01 + (AR0+12) mod BK03), сохранить результат в регистре-аккумуляторе AC3 и загрузить в регистр-указатель AR0 новое значение, равное (AR0+12) mod BK03;

– BSET *(AR7+T0), AR5 – установить в единицу разряд дополнительного регистра AR5 с номером, равным BSA67+(AR7+T0) mod BK47, и загрузить в регистр AR7 новое значение, равное (AR7+T0) mod BK47.

1.1.6.8 Кодирование способов адресации операндов

Для кодирования абсолютной, прямой, косвенной, базовой и индексной адресации операндов используется 7-разрядное поле AAAAAAA:[10]

– 0001000 – код метода абсолютной 16-разрядной адресации памяти, задается операндом вида abs16(#a16);

– 0011000 – код метода абсолютной 23-разрядной адресации памяти, задается операндом вида *(#a23);

– 0101000 – код метода абсолютной 16-разрядной адресации регистра периферийного устройства, задается операндом вида port(#a16);

– 0111000 – код метода косвенной адресации через регистр CDP, задается операндом *CDP;

– 1001000 – код метода косвенной постфиксной автоинкрементной адресации через регистр CDP, задается операндом *CDP+;

– 1011000 – код метода косвенной постфиксной автодекрементной адресации через регистр CDP, задается операндом *CDP−;

– 1101000 – код метода базовой адресации относительно регистра CDP, задается операндом вида *CDP(#K16);

– 1111000 – код метода базовой модифицирующей адресации относительно регистра CDP, задается операндом вида *+CDP(#K16);

– PPP0000 – код метода косвенной адресации через регистр AR0-AR7, задается операндом вида *ARx;

[10] Помимо поля AAAAAAA для задания методов адресации относительно регистра-указателя CDP используется 2-разрядное поле mm, а для задания методов адресации относительно регистров-указателей AR0-AR7 – 3-разрядное поле MMM [45, с. 785].

– PPP0001 – код метода косвенной постфиксной автоинкрементной адресации через регистр AR0-AR7, задается операндом вида *ARx+;

– PPP0010 – код метода косвенной постфиксной автодекрементной адресации через регистр AR0-AR7, задается операндом вида *ARx−;

– PPP0011 – код метода модифицирующей индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T0, задается операндом вида *(ARx + T0);

– PPP0100 – код метода модифицирующей индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T0, задается операндом вида *(ARx – T0);

– PPP0101 – код метода обычной индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T0, задается операндом вида *ARx(T0);

– PPP0110 – код метода базовой адресации относительно регистра AR0-AR7, задается операндом вида *ARx(#K16);

– PPP0111 – код метода модифицирующей базовой адресации относительно регистра AR0-AR7, задается операндом вида *+ARx(#K16);

– PPP1001 – код метода модифицирующей индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T1 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 1 (при установленном флаге ARMS), задаются операндами вида *(ARx+T1) или *ARx(#1);

– PPP1010 – код метода модифицирующей индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T1 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 2 (при установленном флаге ARMS), задаются операндами вида *(ARx–T1) или *ARx(#1);

– PPP1011 – код метода обычной индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T1 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 3 (при установленном флаге ARMS), задаются операндами вида *ARx(T1) или *ARx(#3);

– PPP1100 – код метода косвенной префиксной автоинкрементной адресации через регистр AR0-AR7 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 4 (при установленном флаге ARMS), задаются операндами вида *+ARx или *ARx(#4);

– PPP1101 – код метода косвенной префиксной автодекрементной адресации через регистр AR0-AR7 (при сброшенном флаге ARMS) или код метода базовой

адресации относительно регистра AR0-AR7 со смещением 5 (при установленном флаге ARMS), задаются операндами вида *−ARx или *ARx(#5);

– PPP1110 – код метода бит-реверсивной индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T0 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 6 (при установленном флаге ARMS), задаются операндами вида *(ARx+T0B) или *ARx(#6);

– PPP1111 – код метода бит-реверсивной индексной адресации относительно регистра AR0-AR7 и со смещением в регистре T0 (при сброшенном флаге ARMS) или код метода базовой адресации относительно регистра AR0-AR7 со смещением 7 (при установленном флаге ARMS), задаются операндами вида *(ARx–T0B) или *ARx(#7),
где PPP – поле номера x дополнительного регистра ARx, ARMS – флаг режима косвенной адресации микропроцессора в регистре статуса ST2 (см. пп. 1.1.7.6 на с. 55).

1.1.7 Устройство управления

Устройство управления – часть микропроцессора, формирующая 24-разрядные адреса для чтения кода программы и выполняющая команды ветвления (условные и безусловные переходы, вызовы подпрограмм и возвраты из подпрограмм). Устройство управления состоит из двух частей: регистрового файла и генератора адресов с логикой ветвления (рисунок 1.26).

1.1.7.1 Генератор адресов и логика ветвления

Генератор адресов и логика ветвления связаны:

– с буфером команд (I модуль), откуда поступает код одной или двух параллельно исполняемых команд и непосредственные данные, в них содержащиеся,

– с операционным устройством (D модуль) из регистрового файла которого поступают флаги результата последней выполненной операции, адреса перехода и адреса вызываемых подпрограмм, а также передаются значения регистров устройства управления для записи в регистровый файл операционного устройства;

– с устройством адресации (A модуль), откуда поступает и куда передается содержимое регистров устройства управления, а также поступают флаги результата последней операции устройства адресации;

– с регистровым файлом, откуда поступает и куда передается содержимое управляющих регистров или их полей.

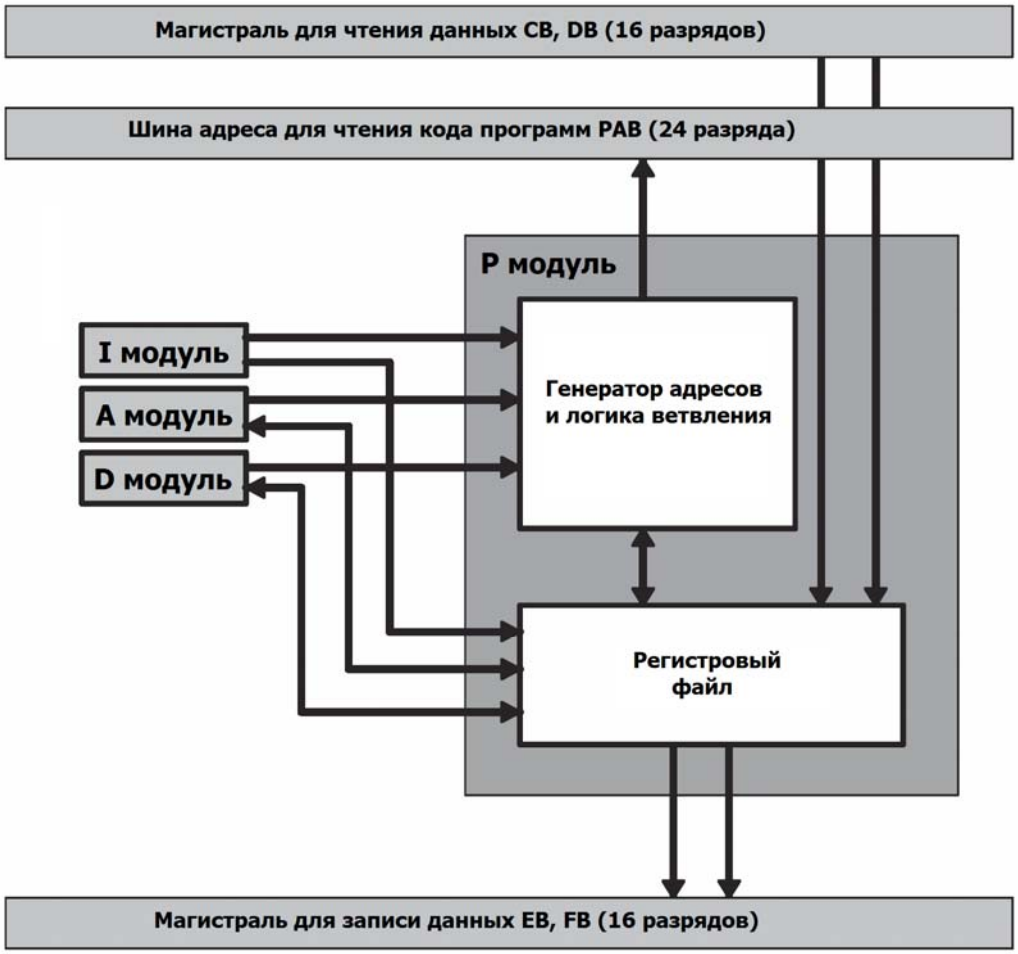


Рисунок 1.26 – Устройство управления

Генератор адресов команд вычисляет 24-разрядные адреса следующей исполняемой команды и передает их на шину адреса PAB. В свою очередь логика ветвления выполняет следующие операции:

– получение от устройства адресации и операционного устройства флагов результата последней операции, которые необходимы для выполнения команд условного ветвления;

– запуск обработчиков прерываний в случае, если пришел запрос на обработку прерывания и прерывание разрешено;

– контроль повторения одиночных команд и блоков команд;

– управление параллельным выполнением двух команд.

1.1.7.2 Регистровый файл

Регистровый файл устройства управления содержит в себе управляющие регистры микропроцессора, содержимое которых может получаться с магистралей для

чтения данных CB и DB, а также передаваться на магистрали для записи данных EB и FB.

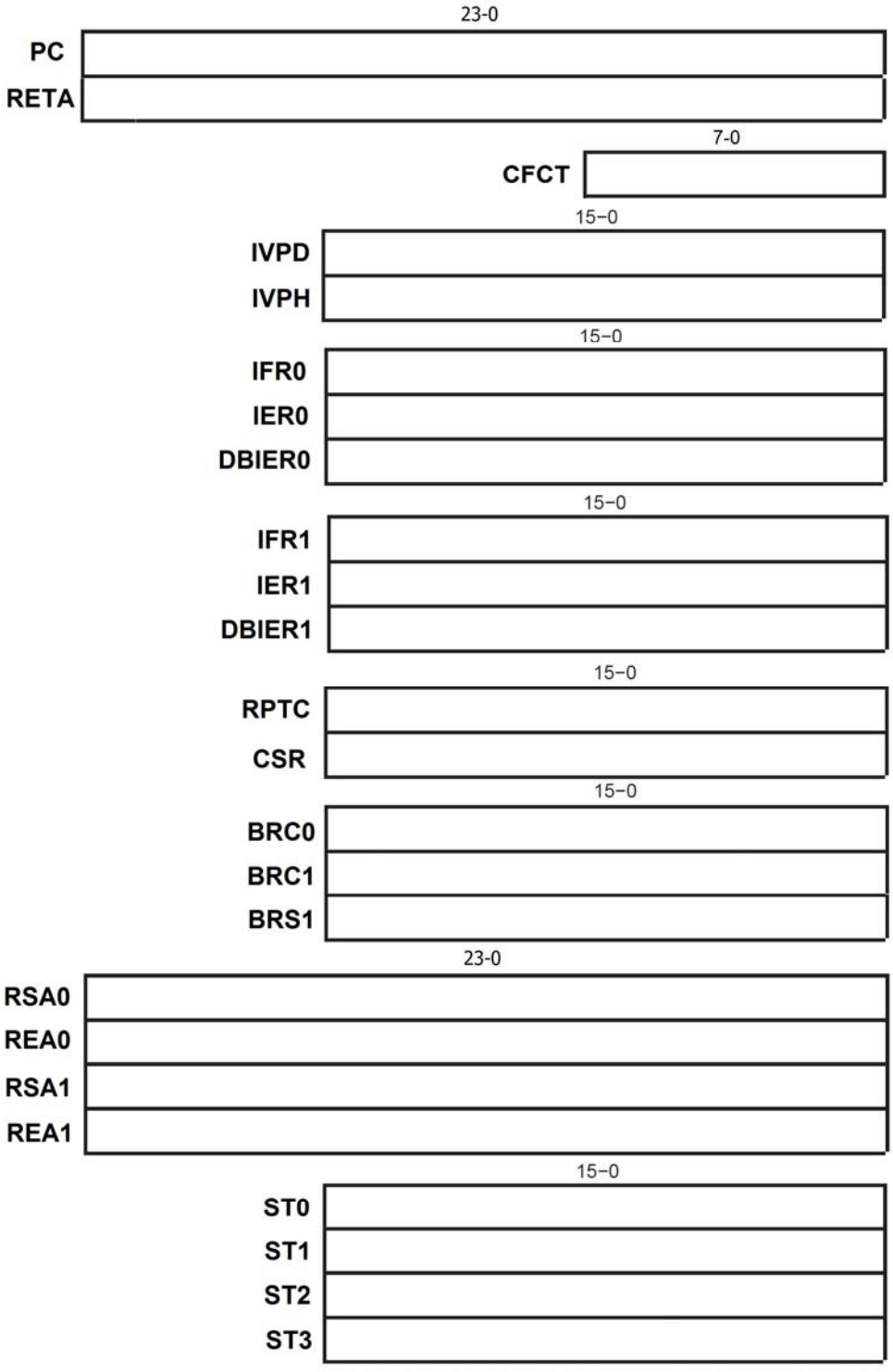


Рисунок 1.27 – Регистры устройства управления

Регистровый файл включает в себя следующие группы управляющих регистров (рисунок 1.27):

1) регистры потока команд:

– PC (англ. Program Counter) – программный счетчик (счетчик команд),

– RETA (англ. Return Address) – регистр адреса возврат,

– CFCT (англ. Control Flow Context) – регистр контекста повторения команд;

2) регистры прерываний:

– IVPD, IVPH (англ. Interrupt Vector Page) – регистры страницы векторов прерываний,

– IFR0, IFR1 (англ. Interrupt Flag Register) – регистры флагов прерываний,

– IER0, IER1 (англ. Interrupt Enable Register) – регистры разрешения прерываний,

– DBIER0, DBIER1 (англ. Debug Interrupt Enable Register) – регистры разрешения отладочных прерываний;

3) регистры повторения одиночной команды:

– RPTC (англ. Repeat Counter) – счетчик повторения команды,

– CSR (англ. Computed Single-Repeat) – регистр числа повторений команды;

4) регистры повторения блоков команд:

– BRC0, BRC1 (англ. Block-Repeat Counter) – счетчики повторения блоков команд,

– BRS1 (англ. Block-Repeat Save) – регистр для хранения начального значения регистра BRC1,

– RSA0, RSA1 (англ. Repeat Start Address) – регистры начальных адресов блоков,

– REA0, REA1 (англ. Repeat End Address) – регистры конечных адресов блоков;

5) регистры статуса:

– ST0-ST3 (англ. Status) – регистры состояние микропроцессора.

1.1.7.3 Регистры потока команд

Регистры потока команд предназначена для контроля хода выполнения команд.

Программный счетчик PC – 24-битный регистр, хранящий адрес первого байта текущей выполняемой команды длиной от 1 до 6 байт, которая декодируется в буфере команд (I модуль). Регистр адрес возврата RETA – это 24-битный регистр, в котором хранится адрес возврата из предыдущей подпрограммы.

В свою очередь регистр контекста повторения команд CFCT – 8-разрядный регистр, в котором хранится состояние циклов повторения одиночной команды и двух блоков команд предыдущей вызванной подпрограммы (рисунок 1.28), где S – флаг повторения одиночной команды, C – флаг условного повторения одиночной команды, B – состояние циклов повторения внешнего блока команд 0 и внутреннего блока команд 1:

0000 – блок 0 неактивный, блок 1 неактивный;

0010 – блок 0 наружный, блок 1 неактивный;

0011 – блок 0 локальный, блок 1 неактивный;

0111 – блок 0 наружный, блок 1 наружный;

1000 – блок 0 наружный, блок 1 локальный;

1001 – блок 0 локальный, блок 1 локальный.[11]

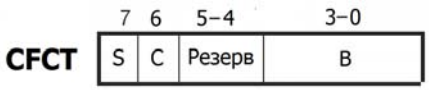


Рисунок 1.28 – Регистр контекста повторения команд

При вызове подпрограмм командами CALL и CALLCC, а также при вызове подпрограмм обработки прерываний командой программного прерывания INT или непосредственно контроллером прерываний, текущее значение PC переписывается в регистр адреса возврата RETA, а текущий контекст повторения команд – в регистр контекста повторения команд CFTC. После этого в регистр PC заносится новое значение – адрес вызываемой подпрограммы и начинается загрузка и выполнение первой команды новой подпрограммы. При возврате из подпрограмм командами RET, RETCC и RETI ранее сохраненное значение регистра PC и текущий контекст повторения команд восстанавливается из регистров RETA и CFTC.

Для обеспечения вложенности вызова подпрограмм перед записью в регистры RETA и CFCT новых значений их старые значения заносится в стеки (рисунок 1.29, а). В свою очередь при возврате из подпрограмм содержимое регистров RETA и CFCT переписывается в программный счетчик PC и в текущий контекст повторения команд, а затем выполняется восстановление из стеков их предыдущих значений.

При вызове подпрограмм обработки прерывания и возврате из таких подпрограмм в стеке дополнительно сохраняются и восстанавливаются регистры статуса ST0-ST2 и регистр состояния отладки DBSTAT (рисунок 1.29, б). [12]

[11] Блок команд является локальным, если он помещается в очередь команд и в процессе повторения команд из этого блока не требует загрузки новых команд из памяти микропроцессора. Блок команд называется наружным, если он не помещается в очередь команд и в процессе повторения команд из этого блока требуется выполнять загрузку буфера команд. Очевидно, внешний блок не может быть локальным, если внутренний блок наружный.

[12] Регистр DBSTAT является программно-недоступным. Не рекомендуется изменять ячейку памяти, из которой восстанавливается значение регистра DBSTAT при возврате из прерывания.

Рисунок 1.29 – Состояние стеков при быстром вызове и возврате

1.1.7.4 Регистры повторения одиночной команды

Регистры повторения одиночной команды RPTC и CSR используются при повторении одиночной или двух команд, выполняемых параллельно. Число повторений команды N загружается в счетчик повторения RPTC до первого ее выполнения. После выполнения команды содержимое счетчика проверяется на ноль. Если значение счетчика не равно нулю, происходит его уменьшение на единицу и процесс выполнения команды повторяется. В итоге одиночная команда повторяется N+1 раз.

Командами, инициирующими повторение одиночной команды, являются команды RPT, RPTCC, RPTADD и RPTSUB. Каждая из этих команд имеется два формата: в первом формате число повторения команды N задается в виде константы в специальном поле команды, во втором формате число повторений команды N предполагается хранящимся в регистре CSR.

1.1.7.5 Регистры повторения блоков команд

Регистры повторения блоков команд BRC0-BRC1, BRS1, RSA0-RSA1 и REA0-REA1 используются для повторения одного блока команд уровня 0 (блок 0) или двух блоков команд, один из которых (внутренний) имеет уровень 1 (блок 1) и вложен в другой (внешний) с уровнем 0 (блок 0).

Командами, инициирующими повторно выполнение блока команд, являются команды RTPB и RPTLOCAL. Команда RPTB начинает выполнение наружного блока команд объемом не более 64 кБ, а команда RPTLOCAL – локального блока команд объемом не более 128 байт. Блок команд начинается сразу за командой повторения и имеет объем, заданный в специальном поле команды.

Когда декодируется команда повторения блока команд, то вначале проверяется, инициировано ли выполнение блока 0. Если выполнение блока 0 уже инициировано, то для организации выполнения блока 1 используются регистры BRC1, RSA1, REA1 и BRS1, в противном случае – BRC0, RSA0 и REA0.

Регистры BRC0, RSA0 и REA0 необходимы для организации выполнения блока 0. Регистр BRC0 является как регистром начального значения числа повторения блока 0, так и счетчиком числа повторений. Как и при повторении одиночной команды, перед выполнением команды повторения в регистр BRC0 загружает число, на единице меньшем, чем требуемое число повторения блока. В свою очередь регистры RSA0 и REA0 содержат 24-разрядные адреса первой и последней команды блока 0 соответственно.

Назначение регистров BRC1, RSA1 и REA1 аналогичное назначению регистров BRC0, RSA0 и REA0. Однако для организации повторения блока 1 требуется регистр BRS1, предназначенный для сохранения начального значения счетчика повторений BRC1. Когда в регистр BRC1 загружается число повторения блока 1, то же значение заносится и в регистр BRS1. В процессе повторения блока 1 происходит модификация счетчика BRC1, однако значение регистра BRS1 не изменяется. По завершению процесса повторения блока 1 содержимое регистра BRS1 переписывается в регистр BRC1. Это позволяет задавать число повторений блока 1 командами, расположенными вне блока 0, что сокращает общее время выполнения вложенных циклов.

1.1.7.6 Регистры статуса

Регистры статуса ST0-ST3 предназначены для контроля состояния микропроцессора и управления режимами его работы (рисунок 1.30), где R (W) – указывает на разрешенную операцию чтения (записи) поля, 0 или 1 – значения разрядов полей после сброса микропроцессора, а на сером фоне показаны поля, используемые в только режиме совместимости с микропроцессорами серии C54.

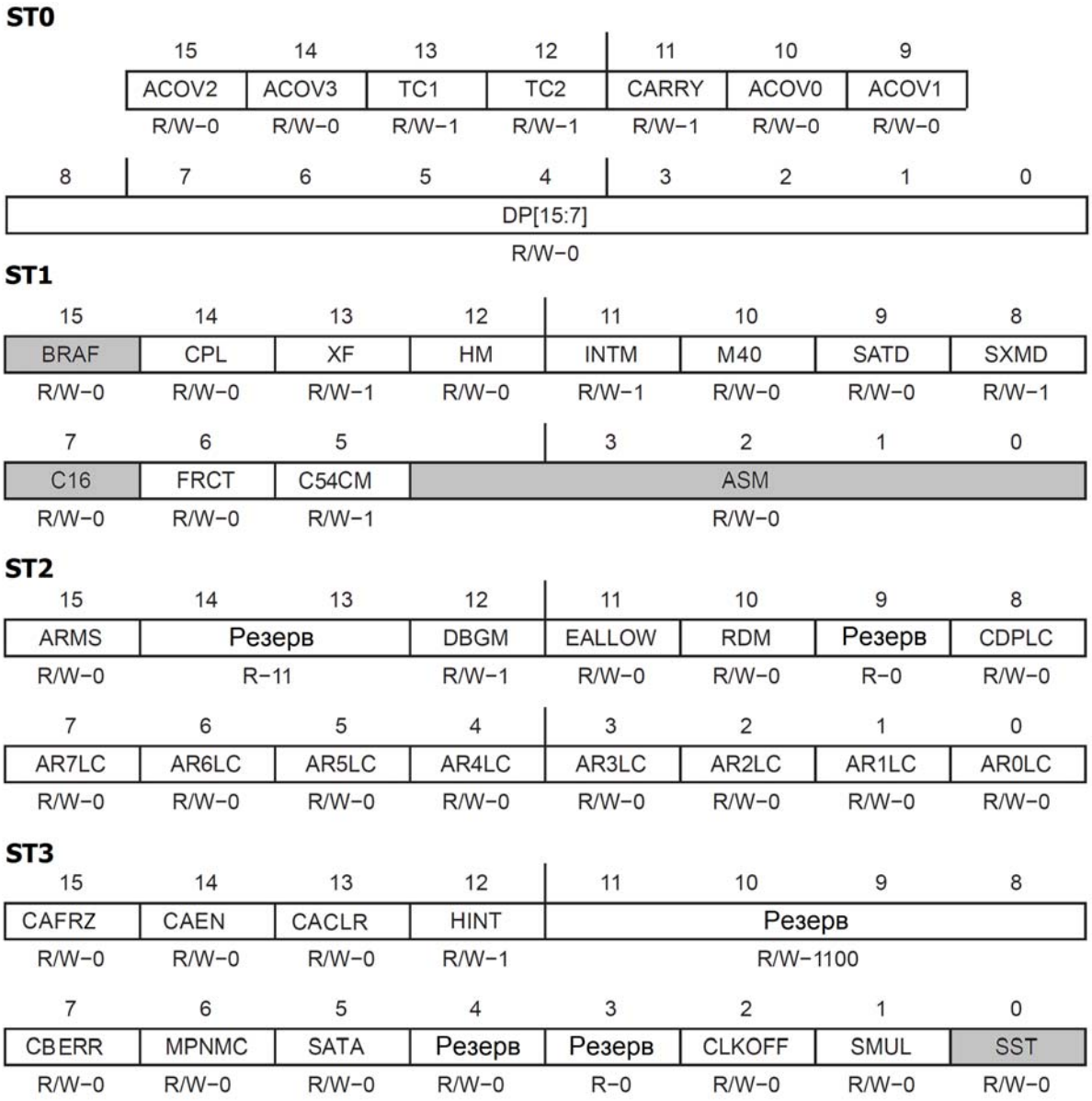


Рисунок 1.30 – Регистры статуса

Регистр ST0 состоит из следующих полей:

– CARRY (англ. Carry) – флаг переноса за пределы разрядной сетки операционного устройства (разрядная сетка устанавливается равной 32 или 40 разрядам в зависимости от состояния флага M40 в ST1);[13]

– ACOV0-ACOV3 (англ. Accumulator Overflow) – флаги переполнения регистров-аккумуляторов AC0-AC3, устанавливаемые при неравенстве переноса из знакового разряда аккумулятора (31 или 39 разряд в зависимости от состояния флага M40 в ST1) переносу в его знаковый разряд;

[13] Флаг переноса при выполнении команд сложения устанавливается равным переносу из знакового разряда, а при выполнении команд сдвига – значению разряда, выдвигаемого за пределы разрядной сетки.

– TC1-TC2 (англ. Test and Control) – флаги проверки и управления, изменяемые некоторыми командами и используемые в командах условных переходов;

– DP[15:7] (англ. Data Page) – теневая копия 9-ти старших разрядов регистра страницы данных DP из регистрового файла устройства адресации, создаваемая для сохранения разрядов с 7 по 15 этого регистра в стеке при возникновении прерываний.

Регистр ST1 состоит из следующих полей:

– C54CM (англ. C54 Compatible Mode) – флаг режима совместимости с микропроцессорами серии C54;[14]

– BRAF (англ. Block-Repeat Active Flag) – флаг активности повторения блока команд (используется только в режиме C54CM);[15]

– C16 (англ. Control 16) – флаг разделения 32-разрядного арифметико-логического устройства D модуля на два независимых и параллельно работающих 16-разрядных арифметико-логических устройства (используется только в режиме C54CM);

– ASM (англ. Accumulator Shift Mode) – поле для числа разрядов сдвига аккумулятора (используется только в режиме C54CM);

– CPL (англ. Compile Mode) – флаг режима компилятора, в котором для прямой адресации операндов используется не регистр страниц данных DP, а регистр указателя стека SS;[16]

– XF (англ. External Flag) – уровень сигнала, устанавливаемый на внешнем выводе микропроцессора XF (0 – низкий уровень, 1 – высокий уровень);

– HM (англ. Hold Mode) – флаг режима ожидания, в котором при поступлении от внешнего интерфейса памяти EMIF сигнала HOLD микропроцессор останавливается, в противном случае – продолжает выполнять программу;

– INTM (англ. Interrupt Mode) – флаг запрета маскируемых прерываний;[17]

– M40 (англ. Mode 40) – флаг 40-разрядного режима работы операционного устройства, если сброшен, то операционное устройство работает в 32-разрядном режиме;

---

[14] Микропроцессоры серии C54 имеют только два 32-разрядных регистра-аккумулятора AC0-AC1, у них нет временных регистров T0-T3 и регистра контекста повторения команд CFCT, а также не реализовано повторение второго блока команд и банковское округление.

[15] У микропроцессоров серии C55 для сохранения флагов активности повторения одиночной команды и блоков команд используется специальный регистр CFCT.

[16] Прямая адресация в пространство ввода-вывода не зависит от состояния флага CPL и всегда выполняется относительно регистра PDP.

[17] Флаг INTM автоматически устанавливается микропроцессором в единицу при вызове процедур обработки прерываний, кроме процедур трассировки, вызываемых командой TRAP. Установка флага INTM не запрещает микропроцессору обработку немаскируемых прерываний.

– SATD (англ. Saturation D) – флаг режима насыщения операционного устройства, в котором при возникновении положительного переполнения результат операции устанавливается равным 007FFFFFFh или 7FFFFFFFFh, а при возникновении отрицательного переполнения – FF80000000h или 8000000000h в зависимости от состояния флага M40;[18]

– SXMD (англ. Sign Extension Mode D) – флаг режима расширения знака, используемый при загрузке аккумуляторов операционного устройства;[19]

– FRCT (англ. Fractional) – флаг дробного режима работы умножителей-аккумуляторов, при нахождении в котором после любого умножения выполняется сдвиг результата на один разряд влево.[20]

Регистр ST2 состоит из следующих полей:

– AR0LC-AR7LC, CDPLC (англ. Linear Circular) – флаги режимов циклической адресации относительно регистров AR0-AR7, CDP;

– RDM (англ. Rounding Mode) – флаг режима округления (0 – математическое округление, 1 – банковское округление);

– EALLOW (англ. Emulation Allow) – флаг разрешения эмулятора (0 – запись в регистры эмулятора запрещена, 1 – запись в регистры эмулятора разрешена);

– DBGM (англ. Debug Mode) – флаг режима откладки (0 – режим отладки разрешен, эмулятор имеет доступ к памяти и регистрам микропроцессора; 1 – режимы отладки запрещен, эмулятор не имеет доступа к памяти и регистрам микропроцессора);

– ARMS (англ. Auxiliary Register Mode Switch) – флаг сигнального режима косвенной адресации операндов.

Регистр ST3 состоит из следующих полей:

– CACLR (англ. Cache Clear) – флаг очистки кэша команд, автоматически сбрасывается после очистки кэша;

– CAEN (англ. Cache Enable) – флаг разрешения кэша команд;

---

[18] Положительное переполнение фиксируется при возникновении переноса в знаковый разряд, равного единице, и из знакового разряда, равного нулю. Отрицательное переполнение – при переносе в знаковый разряд, равном нулю, и из знакового разряда, равном единице.

[19] При расширении знака происходит копирование знакового разряда загружаемого числа во все старшие разряды аккумулятора. Если режим расширения знака выключен, старшие разряды заполняются нулями. Режим расширение знака может также задаваться в специальном поле некоторых команд. В этом случае значение такого поля является более приоритетным по отношению к флагу SXMD.

[20] Все умножители микропроцессора являются целочисленными, т.е. в результате умножения двух целых чисел, равных единице, всегда получается единица. Однако, соответствующее умножение двух дробных чисел, равных 1×2^{-15} в формате Q1.15, дает число 1×2^{-30}, которое в формате Q1.31 интерпретируется как 1×2^{-31} (рисунок 1.8 и 1.9). Следовательно, для коррекции результата умножения дробных чисел необходимо выполнить сдвиг влево на один разряд.

– CAFRZ (англ. Cache Freeze) – флаг блокировки обновления кэша команд (замораживание кэша);[21]

– CBERR (англ. CPU Bus Error) – флаг ошибки передачи данных по внутренним шинам и магистралям микропроцессора, вызывает также установку флага прерывания BERRINTF и автоматически сбрасывается при запуске процедуры обработки прерывания;

– HINT (англ. Host Interrupt) – флаг передачи через интерфейс HPI запроса на обработки прерывания ведущему микропроцессору в многопроцессорной системе обработки сигналов;[22]

– MPNMC (англ. Microprocessor nor Microcomputer) – флаг режима работы внутреннего постоянного запоминающего устройства (0 – режим микрокомпьютера, в котором внутреннее постоянное запоминающее устройство включено; 1 – режим микропроцессора, в котором внутренне постоянное запоминающее устройство отключено);[23]

– SATA (англ. Saturation A) – флаг режима насыщения устройства адресации, в котором при возникновении положительного переполнения результат операции вычисления адреса устанавливается равным 7FFFFh, а при возникновении отрицательного переполнения – 80000h;

– SMUL (англ. Saturation-on-Multiplication) – флаг режима насыщения при умножении (0 – насыщение не выполняется, 1 – при установленных флагах дробного режима FRCT и режима насыщения SATD результат умножения заменяется на 7FFFFFFFh или 7FFFFFFFFh в зависимости от состояния флага M40);[24]

– CLKOFF (англ. Clock Off) – флаг запрета выдачи синхросигнала на выход микропроцессора CLKOUT;

– SST (англ. Saturate-on-Store) – флаг режима насыщения при сохранении аккумулятора, в котором перед сохранением 40-разрядного аккумулятора в 32-разрядной ячейке памяти его содержимое насыщается до 7FFFFFFFh или 80000000h,

[21] У микропроцессора TMS320C5515 кэш команд отсутствует. Кэш команд реализован только у микропроцессоров TMS320C5501, TMS320C5502 и TMS320C5510.
[22] Интерфейс HPI (Host Port Interface) – это 16-разрядный параллельный интерфейс, предназначенный для связи сигнального микропроцессора с ведущим микропроцессором в многопроцессорной системе обработки сигналов. Интерфейс HPI в состав микропроцессора TMS320C5515 не входит. Интерфейс HPI реализован у микропроцессоров TMS320C5501 и TMS320C5502.
[23] У некоторых моделей микропроцессоров начальное значение флага MPNMC зависит от состояния одноименного внешнего вывода. У микропроцессора TMS320C5515 начальное состояние флага MPNMC равно нулю.
[24] Режим насыщения при умножении предназначен для устранения ситуации, при которой в результате умножения двух отрицательных дробных чисел 18000h и 18000h получается положительное число.

если исходное значение аккумулятора больше чем 007FFFFFFh или меньше чем 0080000000h в зависимости от состояния флага SXMD (используется только в режиме C54CM).[25]

1.1.7.7 Регистры прерываний

Прерывание – это временное переключение микропроцессора на выполнение специальной подпрограммы обработки прерывания, которое возникает при поступлении запроса на прерывание от различных источников:

– от схем контроля и управления микропроцессором (сброс микропроцессора, ошибки передачи данных по внутренним шинам и магистралям, и т.п.);

– от контроллеров периферийных устройств (по готовности к передаче данных, при приеме очередной порции данных, при ошибке обмена данными, и т.д.);

– от внешних источников при подаче активных сигналов на входы прерываний микропроцессора;

– при выполнении команд программного прерывания INT и отладочного прерывания TRAP.

Микропроцессор обслуживает 32 источника прерываний, которые разделяются на маскируемые и немаскируемые, аппаратурные и программные, обычные и отладочные. Немаскируемые прерывания вызываются схемами контроля и управления микропроцессором и их нельзя запретить путем сброса соответствующих флагов в регистрах разрешения прерываний. Инициатором аппаратурных прерываний являются сигналы, подаваемыми на входы прерываний микропроцессора или сигналы, выдаваемые контроллерами периферийных устройств. Программные прерывания вызываются командой INT и TRAP, непосредственным операндом которых является номер генерируемого прерывания.

На рисунке 1.31 показаны поля регистров разрешения прерываний IER0 и IER1, регистров флагов прерываний IFR0 и IFR1, а также регистров разрешения отладочных прерываний DBIER0 и DBIER1. В регистрах флагов прерываний фиксируется факт получения запроса на обработку соответствующих прерываний, а в регистрах разрешения прерываний – задаются флаги, разрешающие обработку прерываний. Регистры разрешения отладочных прерываний используется только в режиме эмулятора при отладке программ в реальном масштабе времени.

[25] Вместо флага SST у микропроцессоров серии C55 предусмотрено специальное поля в коде команд, куда заносится значение флага насыщения при сохранении аккумулятора, действующего только во время выполнения команды.

IER0 (IFR0, DBIER0)

15	14	13	12	11	10	9	8
INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
INT7	INT6	INT5	INT4	INT3	INT2	Резерв	
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R–0	

IER1 (IFR1, DBIER1)

15				11	10	9	8
Резерв					RTOSINT	DLOGINT	BERRINT
R–0					R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

Рисунок 1.31 – Регистры прерываний

Регистры IER0 и DBIER0 используются для разрешения обычных и отладочных прерываний с номерами от 2 до 15, а регистр IFR0 содержит флаги прерываний с номерами от 2 до 15. В свою очередь регистры IER1 и DBIER10 используются для разрешения обычных и отладочных прерываний с номерами от 16 до 26, а регистр IFR1 содержит флаги прерываний с номерами от 16 до 26.

Регистры страниц векторов прерываний IVPD и IVPH указывают на номера страниц в памяти программ, где размещаются таблицы векторов прерываний.[26] Вектор прерывания – это описатель подпрограммы обработки прерывания, содержащий адрес точки входа в подпрограмму. В регистре IVPD указывается номер страницы, где размещены векторы прерывания 0-15 и 16-23, а в регистре IVPH – номер страницы, где размещены векторы 16-23 (рисунок 1.32).

Если регистры страниц прерываний имеют одно и то же значение, то они указывают на одну и ту же страницу памяти. Поэтому в странице, на которую указывает регистр IVPH, векторы прерываний 16-23 смещены относительно ее начала на 128 байт.

При аппаратурном сбросе микропроцессора в регистры страниц IVPD и IVPH загружается одно и то же значение, равное 0FFFFh, т.е. при начальном запуске микропроцессора таблица векторов прерываний занимает последнюю страницу постоянного запоминающего устройства.

[26] Объем одной страницы памяти составляет 256 байт, а размер вектора прерываний – 8 байт.

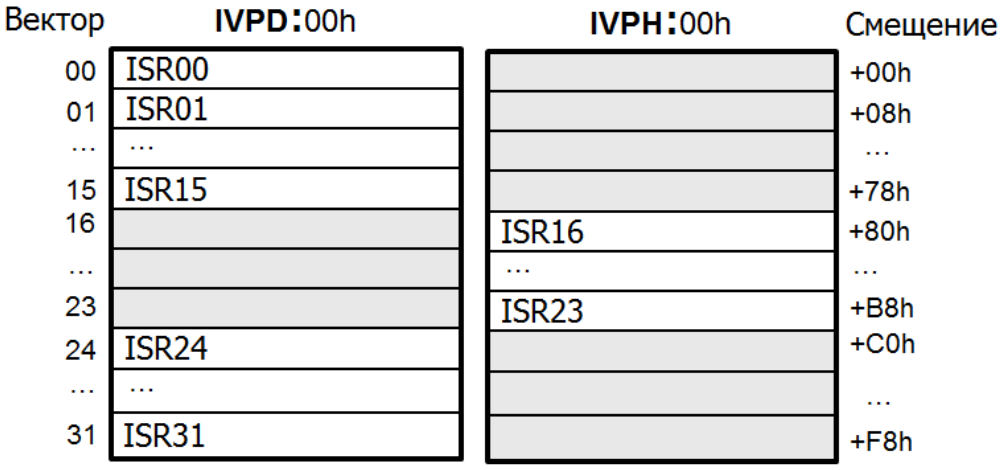


Рисунок 1.32 – Таблицы векторов прерываний

В таблице 1.4 перечислены векторы и соответствующие им источники прерываний, обслуживаемые микропроцессором TMS320C5515.

Таблица 1.4 – Источники прерываний микропроцессора TMS320C5515

Вектор	Адрес	Приоритет	Обозначение и источник прерывания
ISR00	IVPD:00h	1	RESET, по сбросу и инициализации
ISR01	IVPD:08h	3	NMI, от схем контроля (внутреннее немаскируемое)
ISR02	IVPD:10h	5	INT0, внешнее прерывание по входу INT0
ISR03	IVPD:18h	7	INT1, внешнее прерывание по входу INT1
ISR04	IVPD:20h	8	TINT, агрегированное прерывание от таймеров
ISR05	IVPD:28h	9	PROG0, от контроллера I2S0 и MMC/SD0 по передаче
ISR06	IVPD:30h	11	UART, от приемо-передатчика UART
ISR07	IVPD:38h	12	PROG1, от контроллера I2S0 и MMC/SD0 по приему
ISR08	IVPD:40h	13	DMA, от канала прямого доступа к памяти
ISR09	IVPD:48h	15	PROG2, от контроллера I2S1 и MMC/SD1 по передаче
ISR10	IVPD:50h	16	FFT, от аппаратного ускорителя БПФ
ISR11	IVPD:58h	17	PROG3, от контроллера I2S1 и MMC/SD1 по приему
ISR12	IVPD:60h	19	LCD, от контроллера дисплея
ISR13	IVPD:68h	20	SAR, агрегированное от АЦП
ISR14	IVPD:70h	23	XTM2, от контроллера I2S2 по передаче

Таблица 1.4 – Источники прерываний микропроцессора TMS320C5515

Вектор	Адрес	Приоритет	Обозначение и источник прерывания
ISR15	IVPD:78h	24	RCV2, от контроллера I2S2 по приему
Вектор	Адрес	Приоритет	Обозначение и источник прерывания
ISR16	IVPH:80h	6	XMT3, от контроллера I2S3 по передаче
ISR17	IVPH:88h	10	RCV3, от контроллера I2S3 по приему
ISR18	IVPH:90h	14	RTC, от часов реального времени
ISR19	IVPH:98h	18	SPI, от контроллера SPI
ISR20	IVPH:A0h	21	USB, от контроллера USB
ISR21	IVPH:A8h	22	GPIO, от портов ввода-вывода общего назначения
ISR22	IVPH:B0h	23	EMIF, от интерфейса внешней памяти при ошибках
ISR23	IVPH:B8h	26	I2C, от контроллера I2C
ISR24	IVPD:C0h	4	BERR, при ошибке обмена по шинам и магистралям
ISR25	IVPD:C8h	27	DLOG, протоколирование данных (программное)
ISR26	IVPD:D0h	28	RTOS, вызов операционной системы (программное)
ISR27	IVPD:D8h	–	RTDRCV, от эмулятора по приему
ISR28	IVPD:E0h	–	RTDXMT, от эмулятора по передаче
ISR29	IVPD:E8h	2	EMUINT, от монитора эмулятора
ISR30	IVPD:F0h	–	SINT30, пользовательское (программное)
ISR31	IVPD:F8h	–	SINT31, пользовательское (программное)

Структура вектора прерывания ISR (англ. Interrupt Service Routine) показана на рисунке 1.33, где SC (англ. Stack Configuration) – байт конфигурации стека микропроцессора, EP (англ. Entry Point) – точка входа, или 24-разрядный адрес процедуры обработки прерывания. Байт конфигурации стека принимает следующие значения:

00h – быстрый вызов-возврат и два независимых 16-разрядных стека;

10h – медленный вызов-возврат и два независимых 16-разрядных стека;

20h – медленный вызов-возврат и единый 32-разрядный стек.

При медленном вызове подпрограмм и медленном возврате из подпрограмм в отличие ранее описанного быстрого вызова и возврата (см. пп. 1.1.7.3) регистры адреса возврата RETA и контекста повторения команд CFCT не используются. В стеки непосредственно записывается и из стека непосредственно считывается содержимое программного счетчика PC и текущего состояние контекста повторения команд (рисунок 1.34).

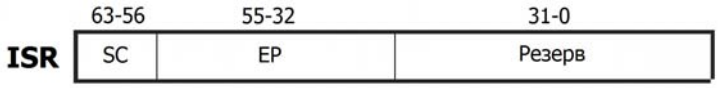


Рисунок 1.33 – Вектор прерывания

Такие операции требуют большего времени, так как переход к выполнению вызываемой подпрограммы задерживается до окончания операции сохранения текущих значений программного счетчика PC и контекста повторения в стеке, а возврат из подпрограммы – задерживается до окончания операции восстановления старых значений программного счетчика PC и контекста повторения команд.

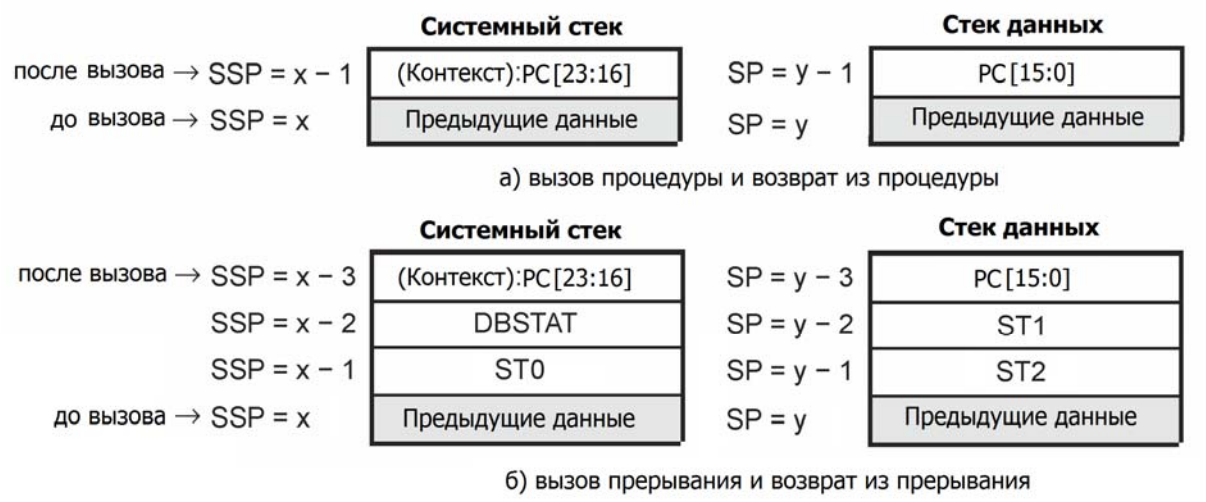


Рисунок 1.34 – Состояние стеков при медленном вызове и возврате

При быстром вызове переход к выполнению вызываемой подпрограммы осуществляется параллельно с записью в стек содержимого регистров адреса возврата RETA и контекста повторения команд CFTC, а при быстром возврате – возврат в точку вызова подпрограммы осуществляется параллельно с восстановлением предыдущего содержимого регистров RETA и CFTC из стека. Условно выполнение быстрого вызова подпрограммы и быстрого возврата из подпрограммы можно описать так:

[PC, Контекст повторения] → [RETA, CFCT] **::** [RETA, CFCT] → [*–SP, *–SSP];

[RETA, CFCT] → [PC, Контекст повторения] **::** [*SP–, *SSP–] → [RETA, CFCT],

где двойное двоеточие обозначает параллельное выполнение операций левой и правой частей вызова подпрограммы CALL (первая формула) и возврата из подпрограммы RET (вторая формула).

Помимо быстрого и медленного вызова и возврата из подпрограмм в векторе прерываний может задаваться конфигурация стеков, действующая при выполнении подпрограммы обработки прерывания. Использование двух раздельных 16-разрядных стеков было описано ранее. При конфигурировании единого 32-разрядного стека значение регистра SSP изменяется так же, как и значение регистра SP, например, при записи в стек с использованием регистра SP содержимое регистра SSP также уменьшается, как и уменьшается содержимое регистра SP, а при чтении из стека – содержимое регистров SP и SSP увеличивается одинаковым образом.

1.1.7.8 Методы адресации команд

Адресация команд служит для изменения последовательности выполнения команд и используется в командах условного и безусловного перехода, а также в командах условного и безусловного вызова подпрограмм.

Для адресации команд перехода или первой команды подпрограммы используются следующие поля в форматах соответствующих команд:

– поле SS – 2-разрядное поле с номером регистра-аккумулятора AC0-AC3, в котором содержится 24-разрядный адрес команды;

– поле l4 – 4-разрядное поле беззнакового смещения адресуемой команды относительно текущего значения счетчика команд PC;

– поле L7 – 7-разрядное поле знакового смещения адресуемой команды относительно текущего значения счетчика команд PC;

– поле L8 – 8-разрядное поле знакового смещения адресуемой команды относительно текущего значения счетчика команд PC;

– поле L16 – 16-разрядное поле знакового смещения адресуемой команды относительно текущего значения счетчика команд PC;

– поле P24 – 24-разрядное поле адреса команды.

Безусловные переходы в программе осуществляются с помощью команды B (англ. Branch):

– B ACx – безусловный переход по адресу в регистре-аккумуляторе ACx (формат команды 1001 0001 0000 00SS);

– B L7 – безусловный переход по адресу PC+L7 (формат команды 0100 101E 0LLL LLLL);

– B L16 – безусловный переход по адресу PC+L16 (формат команды 0000 011E LLLL LLLL LLLL LLLL);

– B P24 – безусловный переход по адресу P24 (формат команды 0110 1010 PPPP PPPP PPPP PPPP PPPP PPPP);

а условные – с помощью команды BCC (англ. Branch Conditionally):

– BCC l4, cond – переход по адресу PC+l4 при выполнения условия cond (формат команды 0110 0111 1CCC CCCC);

– BCC L8, cond – переход по адресу PC+L8 при выполнения условия cond (формат команды 0000 010E 0CCC CCCC LLLL LLLL);

– BCC[U] L8, src RELOP K8 – переход по адресу PC+L8 при выполнения условия src RELOP K8 (формат команды);

– BCC L16, cond – переход по адресу PC+L16 при выполнения условия cond (формат команды 0110 1101 0CCC CCCC LLLL LLLL LLLL LLLL);

– BCC L16, ARy_mod != #0 – переход по адресу PC+L16 при выполнения условия ARy_mod != #0 (формат команды 1111 1100 AAAA AAAI LLLL LLLL LLLL LLLL);

– BCC P24, cond – переход по адресу P24 при выполнения условия cond (формат команды 0110 1000 0CCC CCCC PPPP PPPP PPPP PPPP PPPP PPPP),

где:

x – номер регистра-аккумулятора (0, 1, …, 3), кодируемый в поле SS;

E – флаг параллельного выполнения команды;

cond – проверяемое условие, кодируемое 7-разрядным полем CCC CCCC (см. таблицу 1.5);

U – флаг беззнакового сравнения;

ARy_mod – адрес операнда-источника, кодируемый полем AAAA AAAI (см. пп. 1.1.6.8 на с. 47) с участием регистра ARy, значение которого после вычисления адреса операнда сравнивается с нулем;

y – номер дополнительного регистра (0, 1, …, 7).

Таблица 1.5 – Кодирование поля CCC CCCC

Код	Описание проверяемого условия
000 FSSS	src == 0 – проверка на равенство нулю содержимого регистра src
001 FSSS	src != 0 – проверка на не равенство нулю содержимого регистра src
010 FSSS	src < 0 – проверка отрицательность содержимого регистра src
011 FSSS	src <= 0 – проверка на не равенство нулю регистра src
100 FSSS	src > 0 – проверка на не равенство нулю регистра src
101 FSSS	src >= 0 – проверка на не равенство нулю регистра src
110 00SS	overflow(ACx) – проверка установки флага переполнения ACOVx
110 0100	TC1 – проверка установки флага TC1
110 0101	TC2 – проверка установки флага TC2
110 0110	CARRY – проверка установки флага переноса CARRY
110 0111	Резерв
110 1000	TC2 & TC2 – проверка установки флагов TC1 и TC2
110 1001	TC2 & !TC2 – проверка установки флага TC1 и сброса флага TC2
110 1010	!TC2 & TC2 – проверка сброса флага TC1 и установки флага TC2
110 1011	!TC2 & !TC2 – проверка сброса флагов TC1 и TC2
110 11xx	Резерв
111 00SS	!overflow(ACx) – проверка сброса флага переполнения ACOVx
111 0100	!TC1 – проверка сброса флага TC1
111 0101	!TC2 – проверка сброса флага TC2
111 0110	!CARRY – проверка сброса флага переноса CARRY
111 0111	Резерв
111 1000	TC2 | TC2 – проверка установки флагов TC1 или TC2
111 1001	TC2 | !TC2 – проверка установки флага TC1 или сброса флага TC2
111 1010	!TC2 | TC2 – проверка сброса флага TC1 или установки флага TC2
111 1011	!TC2 | !TC2 – проверка сброса флагов TC1 или TC2
111 1100	TC2 ^ TC2 – проверка не равенства флагов TC1 и TC2
111 1101	TC2 ^ !TC2 – проверка равенства флагов TC1 и TC2
111 1110	!TC2 ^ TC2 – проверка равенства флагов TC1 и TC2
111 1111	!TC2 ^ !TC2 – проверка не равенства флагов TC1 и TC2

Примеры переходов в программе:

– B $+#12 – выполнить переход на команду с адресом PC+12, где $ – знак счетчика команд PC, а для кодирования команды используется формат с полем l4;

– B $–#12 – выполнить переход на команду с адресом PC–12, где $ – знак счетчика команд PC, а для кодирования команды используется формат с полем L7;

– BCC Label, AR0 != 0 – выполнить переход на команду с меткой Label при условии равенства содержимого регистра AR0 нулю, где в зависимости от смещения адресуемой команды относительно счетчика команд могут использоваться команды с полями l4, L8, L16 или P24.

В свою очередь безусловные вызовы подпрограмм осуществляются с помощью команды CALL (англ. Call):

– CALL ACx – вызов подпрограммы с адресом в регистре-аккумуляторе ACx (формат команды 1001 0010 0000 00SS);

– CALL L16 – вызов подпрограммы с адресом PC+L16 (формат команды 0000 100E LLLL LLLL LLLL LLLL);

– CALL P24 – вызов подпрограммы с адресом P24 (формат команды 0110 1100 PPPP PPPP PPPP PPPP PPPP PPPP),

а условные – с помощью команды CALLCC (англ. Call Conditionally):

– CALL L16, cond – вызов подпрограммы с адресом PC+L16 при выполнении условия cond (формат команды 0110 1110 0CCC CCCC LLLL LLLL LLLL LLLL);

– CALL P24, cond – вызов подпрограммы с адресом P24 при выполнении условия cond (формат команды 0110 1001 0CCC CCCC PPPP PPPP PPPP PPPP PPPP PPPP),

где:

x – номер регистра-аккумулятора (0, 1, …, 3), кодируемый в поле SS;

E – флаг параллельного выполнения команды;

cond – проверяемое условие, кодируемое 7-разрядным полем CCC CCCC (см. таблицу 1.5).

Примеры вызова подпрограмм:

– CALL AC1 – вызов подпрограммы с адресом в регистре-аккумуляторе AC1;

– CALL Label, overflow(AC3) – вызов подпрограммы с меткой Label при взведенном флаге переполнения регистра-аккумулятора AC3, где в зависимости от смещения адресуемой подпрограммы относительно счетчика команд PC могут использоваться команды с полями L16 или P24.

1.2 Средства разработки и отладки программ

Для разработки программ в среде CCS предусмотрен широкий спектр средств (рисунок 1.35). Это компилятор, ассемблер, архиваторы для создания макробиблиотек и библиотек объектных модулей, компоновщик, библиотекарь и т.д.

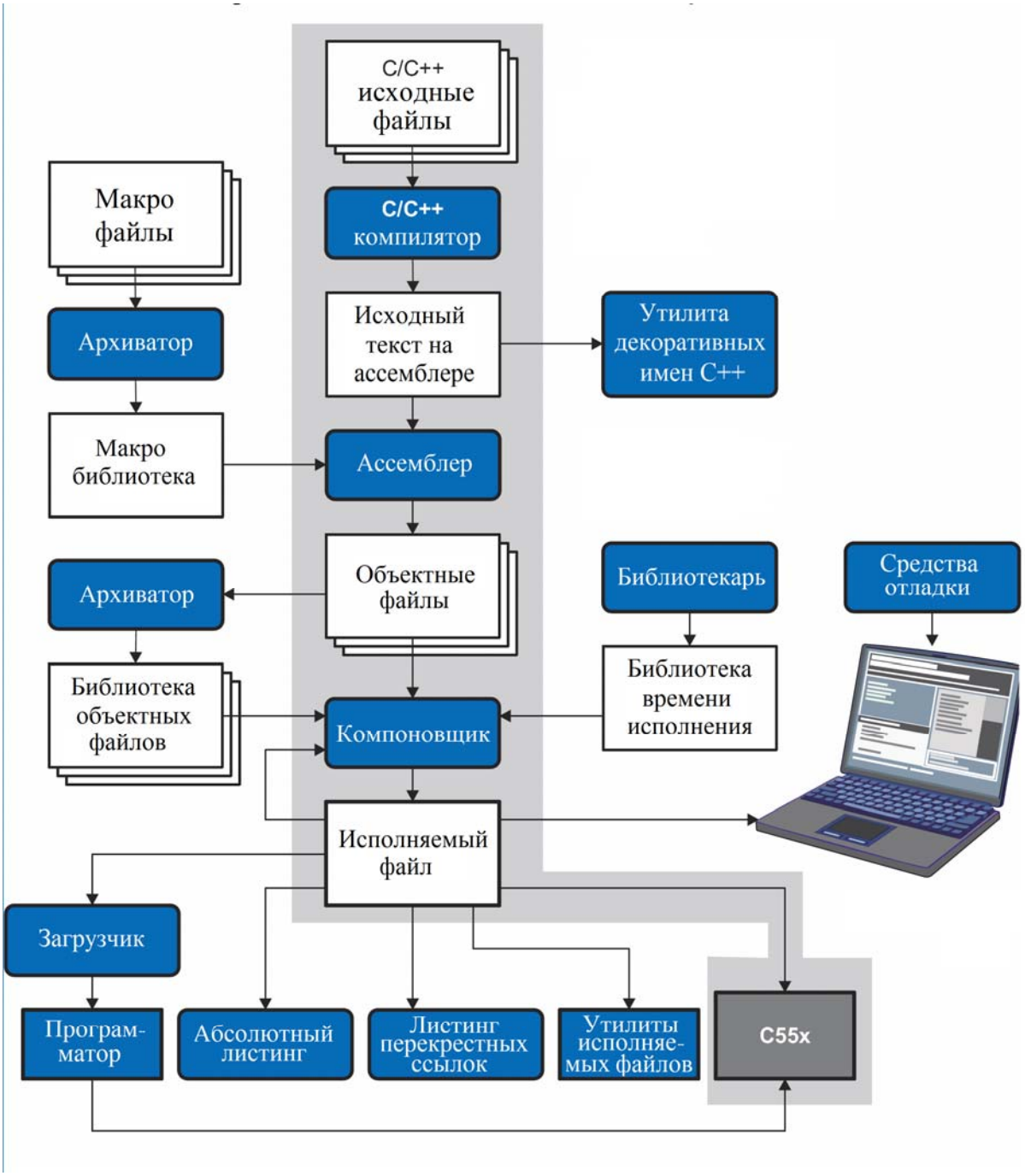


Рисунок 1.35 – Средства разработки и отладки программ

Компилятор в среде CCS порождает не машинный код, а тексты на языке ассемблера, которые на следующем этапе ассемблером транслируются в объектные файлы. Созданные ассемблером объектные файлы и ранее подготовленные с помощью архиватора объектные библиотеки объединяются компоновщиком в единый исполняемый файл, который может быть загружен в память микропроцессора для исполнения.

1.2.1 Компилятор C и C++

При выполнении программ, откомпилированных с помощью компилятора с языка программирования C и C++, микропроцессор переводится в состояние, описанное в таблице 1.6 [44, с. 124], где в последней колонке указано, выполняет ли компилятор модификацию соотвествующих флагов и каково значение этих флагов по умолчанию.

Таблица **1**.6 – Состояние микропроцессора для программ на языке C и C++

Флаги	Описание	Модификация
ST0 ACOV[0-3]	Переполнение аккумуляторов AC0-AC3	да
ST0 CARRY	Перенос	да
ST0 TC[1-2]	Проверяемые условия TC1-TC2	да
ST0 DP[7-15]	Копия регистра страниц данных DP	нет
ST1 CPL	Режим компилятора	нет, CPL=1
ST1 INTM	Маскирование прерываний	нет, INTM=0
ST1 M40	Разрядность операционного устройства	да, M40=0
ST1 SATD	Насыщение операционного устройства	да, SATD=0
ST1 SXMD	Расширения знака операционного устройства	нет, SXMD=1
ST1 FRCT	Сдвиг влево при сохранении результата	да, FRCT=0
ST2 ARMS	Режим косвенной адресации через ARx	нет, ARMS=1
ST2 RDM	Округление результата операций	да, RDM=0
ST2 CDPLC	Циклическая адресация через регистр CDP	да, CDPLC= 0
ST2 AR[0-7]LC	Циклическая адресация через регистры ARx	да, ARxLC=0

Таблица **1**.6 – Состояние микропроцессора для программ на языке С и С++

Флаги	Описание	Модификация
ST3 MPNMC	Микропроцессор-микрокомпьютер	нет, MPNMC=0
ST3 SATA	Насыщение устройства адресации	да, SATA=0
ST3 SMUL	Насыщение при умножении	да, SMUL=1
ST3 SST	Насыщение при сохранении аккумуляторов	нет, SST=0

Ниже приведены отличия реализации языка С и С++ в CCS [44] по отношению к стандартным реализациям [10, 23], которые связаны с необходимостью разработки эффективных программ для широкого спектра архитектур микропроцессоров.

1.2.1.1 Встроенные типы данных

Встроенные типы данных языков С, С++ и их характеристики в CCS приведены в таблице 1.7 [44, с. 84]. При разработке программ необходимо следить за соответствием разрядности переменных и диапазонами представления чисел.

Таблица **1**.7 – Типы данных языка С и С++

Тип	Размер	Формат	Минимум	Максимум
char, signed char	16	ASCII	-32768	32767
unsigned char	16	ASCII	0	65535
short, signed short	16	знаковый	-32768	32767
unsigned short	16	без знака	0	65535
int, signed int	16	знаковый	-32768	32767
unsigned int	16	без знака	0	65535
long, signed long	32	знаковый	-2147483648	2147483647
unsigned long	32	без знака	0	4929967295
float	32	IEEE	1,175494e-38	3,40282346e+38
double	32	IEEE	1,175494e-38	3,40282346e+38

Таблица **1**.7 – Типы данных языка C и C++

Тип	Размер	Формат	Минимум	Максимум
long double	32	IEEE	1,175494e-38	3,40282346e+38
long long, signed long long	40	знаковый	-549755813888	549755813887
unsigned long long	40	без знака	0	1099511627775
перечисление enum (C)	16	знаковый	-32768	32767
перечисление enum (C++)	16, 32, 40	знаковый	-549755813888	549755813887
указатели данных	16, 23	адресный	0	0x7FFFFF
указатели функций	24	адресный	0	0xFFFFFF

При размещении данных в памяти компилятор выполняет выравнивание переменных и полей по границе слов и двойных слов (рисунок 1.36).

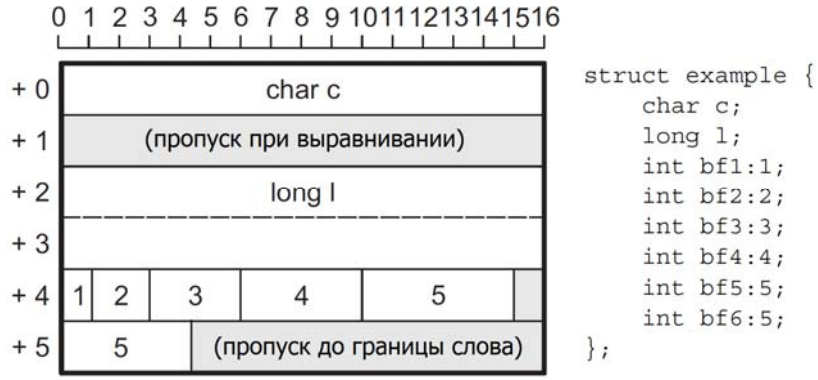


Рисунок 1.36 – Средства разработки и отладки программ

1.2.1.2 Вызов функций

Передача аргументов функции осуществляется в порядке их перечисления в объявлении согласно следующим правилам [44, с. 125]:

1) аргументы передаются в регистрах микропроцессора и через стек в следующем порядке:

– 16-разрядные указатели – в регистрах AR0, AR1, …, AR4;

– 24-разрядные указатели – в регистрах XAR0, XAR1, …, XAR4;

– 16-разрядные данные – в регистрах T0, T1, AR0, …, AR4;

– 32-разрядные и 40-разрядные данные – в регистрах AC0, AC1, AC2;

– при переменном числе аргументов (в объявлении функции используется многоточие в качестве последнего аргумента) аргумент перед многоточием и все последующие аргументы, заданные вместо многоточия, загружаются в стек;

– через стек передаются те аргументы, которые не поместились в регистры, выделенные для передачи аргументов;

2) аргументы-структуры объемом более 32 бита передаются в виде указателя на область памяти, а менее 32 бит – в регистрах соответствующей разрядности;

3) загрузка аргументов в стек осуществляется в порядке, обратном их записи в объявлении функции, например:

SP(0) и XSP(0) – адрес возврата из функции;

SP(1) – первый 16-разрядный аргумент;

SP(2) – второй 16-разрядный аргумент;

SP(3) – пропуск слова для выравнивания по границе двойного слова;

SP(4) – первое слово третьего 32-разрядного аргумента;

SP(5) – второй слово третьего 32-разрядного аргумента и т.д.,

где 24-разрядные аргументы-указатели, 32-разрядные и 40-разрядные аргументы-данные выравниваются в стеке по границе двойного слова, при котором смещение аргумента относительно указателя стека SP кратно двум (задано в круглых скобках).

Доступ к данным в стеке осуществляется посредством прямой адресации через регистр-указатель SP, что возможно только при установленном флаге CPL в регистре статуса ST1. Если флаг CPL сброшен, то при прямой адресации вместо регистра SP микропроцессора используется регистр-указатель DP.

Возвращаемое значение функции передается:

– 16-разрядные данные – в регистре T0;

– 16-разрядный указатель – в регистре AR0;

– 24-разрядный указатель – в регистре XAR0;

– 32-разрядные и 40-разрядные данные – в регистре AC0;

– структура объемом до 32 бит – в регистре AC0;

– структура объемом более 32 бит загружается в область памяти, выделяемой вызывающей функцией в своем стеке, и указатель на которую передается в качестве первого аргумента функции.

1.2.1.3 Сохранение и восстановление регистров

В теле функции регистры, зарезервированные для передачи аргументов, могут изменяться произвольным образом. Однако другие регистры, например, T2-T3, XAR5-

XAR7, AC3, ST1-ST4, если они модифицируется в теле функции, должны быть вначале сохранены в стеке, а после их использования – восстановлены. Если в теле функции осуществляется вызов других функций, то для предотвращения потери данных в регистрах-аргументах T0-T1, XAR0-XAR4 и AC0-AC3 их необходимо сохранять.

Вызывающей функцией при использовании также сохраняются регистры циклической адресации BK03, BK47, BKC, BSA01, BSA23, BSA45, BSA67, BSAC, регистры простого и блочного повторения RPTC, CSR, BRC0, BRC1, RSA0, RSA1, REA0, REA1, BRS1, регистр-указатель коэффициентов XCDP и регистры передачи TRN0-TRN1.

1.2.1.4 Локальная память

Выделение и освобождение локальной памяти в теле функции осуществляется уменьшением и увеличением указателя стека SP на число выделяемых слов, например,

AADD #–K, SP; выделение локальной памяти в объеме K слов,

AADD #K, SP; освобождение локальной памяти в объеме K слов.[27]

После выделения локальной памяти смещения аргументов функции относительно регистра-указателя SP увеличиваются на число выделенных слов K.

1.2.1.5 Обработка прерываний

Для определения на языке C функций – обработчиков прерываний, используется квалификатор interrupt [44, с. 85]. Функции, описанные с квалификатором interrupt, не возвращают значений и не принимают аргументов. При входе в тело такой функции все регистры, используемые компилятором, сохраняются в стеке, а при выходе – восстанавливаются. Компилятором также предполагается, что состояние микропроцессора при вызове функции может отличаться от состояния, описанного в таблице **1**.6. Для возврата из функции вместо команды возврата из подпрограммы RET используется команда возврата из прерывания RETI.

Для указания на функцию обработки прерываний также могут использоваться специальные управляющие директивы препроцессора [44, с. 101]:

#pragma INTERRUPT [(***имя***)];

#pragma INTR_FUNC ([***имя***,] ***вектор***),

где ***имя*** – идентификатор функции обработки прерывания, если имя не задано, предполагается следующая после директивы функция; ***вектор*** – номер вектора прерывания в диапазоне от 0 до 31. В отличие от первой директивы, которая

[27] Команда AADD, в отличие от команды ADD, выполняется устройством адресации на стадии вычисления адресов операндов.

эквивалентна по действию квалификатору interrupt, вторая директива предназначена для указания на функцию обработки программных прерываний, вызываемую не командой вызова подпрограммы CALL, а командой программного прерывания INTR. Функции программных прерываний используются для вызова операционных систем и могут возвращать значения и принимать аргументы.

Следует обратить внимание на то, что для всех типов функций обработки прерываний компилятор не порождает код, создающий и записывающий в память соответствующий вектор обработки прерываний. Задание векторов обработки прерываний должно быть реализовано или с помощью команд компоновщика, или с помощью специальной программы.

1.2.1.6 Порты ввода-вывода

Для размещения глобальных и статических переменных в адресном пространстве ввода-вывода микропроцессора используется квалификатор ioport [44, с. 86]. Указатели с квалификатором ioport являются 16-разрядными, так как адресное пространство ввода-вывода имеет размер 64 килослова.

1.2.1.7 Квалификатор onchip

Квалификатор onchip сообщает компилятору о размещении данных во внутренней памяти микропроцессора [44, с. 88]. Это позволяет использовать такие данные для двойных операций умножения в умножителях-аккумуляторах операционного устройства. В противном случае, при размещении этих данных во внешней памяти, будет возникать ошибка доступа к данным.

1.2.1.8 Перекрытие областей памяти

Квалификатор ограничения перекрытия данных в памяти restrict специфицирует указатель на область памяти, которая рассматривается компилятором как не имеющая перекрытия с другими областями памяти, также выделенными для хранения данных и заданных указателями. Это позволяет компилятору исключить генерацию кода, выполняющего проверку на перекрытие областей памяти и изменения порядка обращения к данным внутри перекрывающихся областей при их модификации.

1.2.1.9 Изменчивые данные

Квалификатор изменчивых данных volatile отключает оптимизацию доступа к ячейкам памяти [44, с. 89]. При описании переменной с квалификатором volatile компилятор предполагает, что значение переменной может измениться в произвольный момент времени неконтролируемым образом.

1.2.1.10 Ассемблерные вставки

Оператор ассемблирования asm используется для прямой вставки в программу на языке C и C++ операторов ассемблера [44, с. 91]:

asm("***текст***"),

где ***текст*** – оператор ассемблера, заданный в виде строки по правилам языка C (возможно с управляющими последовательностями).

1.2.1.11 Секционирование программы

Директивы секционирования служат для размещения данных или кода программ в именованных секциях [44, с. 92, 105]:

#pragma CODE_SECTION ([***имя***,] "***имя-секции***");

#pragma DATA_SECTION ([***имя***,] "***имя-секции***");

#pragma SET_CODE_SECTION ("***имя-секции***");

#pragma SET_DATA_SECTION ("***имя-секции***"),

где ***имя*** – идентификатор функции или переменной, ***имя-секции*** – идентификатор секции кода или данных. Если имя функции или переменной не задано, то директива действует на следующее за ней определение.

1.2.1.12 Выравнивание данных

Директива выравнивания данных [44, с. 96]

#pragma DATA_ALIGN([***имя***,] ***константа***)

используется для размещения переменной ***имя*** по адресу, кратному числу ***константа***. Если ***имя*** не задано, то директива действует на следующее после нее объявление. Значение константы должно быть кратно степени двойки.

1.2.1.13 Прямая подстановка

Для управления прямой подстановкой кода тела функции вместо ее вызова используются следующие директивы [44, с. 98]:

#pragma FUNC_ALWAYS_INLINE [(***имя***)];

#pragma FUNC_CANNOT_INLINE [(***имя***)],

где ***имя*** – идентификатор функции. Если имя не задано, то предполагается функция, непосредственно следующая за директивой. Действие первой директивы эквивалентно квалификатору inline в языке программирования C++. Вторая директива запрещает прямую подстановку тела функции.

1.2.1.14 Побочные эффекты

Для сообщения компилятору особенностей определения функций используются следующие директивы [44, с. 99]:

#pragma FUNC_IS_PURE [(***имя***)];

#pragma FUNC_NEVER_RETURNS [(***имя***)];

#pragma FUNC_NO_GLOBAL_ASG [(***имя***)];

#pragma FUNC_NO_IND_ASG [(***имя***)].

Директива FUNC_IS_PURE применяется для функций, не имеющих побочных эффектов (о побочных эффектах см. [10, с. 75-76, с. 121, с. 254, с. 257]). Это позволяет компилятору удалить вызов функции, если ее возвращаемое значение не используется, а также удалить дополнительные вызовы функции при неизменных аргументах. Директива FUNC_NEVER_RETURNS необходима для функций, не возвращающих управление в точку вызова. В свою очередь директива FUNC_NO_GLOBAL_ASG (FUNC_NO_IND_ASG) специфицирует функции, которые не присваивают значения глобальным переменным (не присваивают значения переменных с помощью указателей) и не имеет в своем теле оператора asm.

1.2.1.15 Оптимизация циклов

Директива итерации [44, с. 103]

#pragma MUST_ITERATE(***минимум***, ***максимум***, ***кратность***)

сообщает компилятору минимальное и максимальное число повторений циклов for, while или do, а также кратность изменения переменной цикла. Эти данные позволяют компилятору сгенерировать оптимальный код для тела циклов.

Директива развертывания [44, с. 105]

#pragma UNROLL (***константа***)

используется для оптимизации циклов путем сокращения числа итераций и сообщает компилятору число копий тела цикла, которые требуется сделать. Например, цикл

```
for( i = len; i >= 0; i++) {
     a[i] = fun(i);
}
```

после двукратного развертывания будет эквивалентен циклу

```
for( i = len; i >= 0; i+=2) {
     a[i] = fun(i);
     a[i+1] = fun(i+1);
}.
```

1.2.2 Ассемблер

Программа на ассемблере [43] состоит из строк, в каждой из которых записывается один оператор вида

[***метка***[:]] ***пробел мнемоника*** [***операнд***[, ***операнд*** …]] [; ***комментарий***],

где ***метка*** – ссылочное имя оператора (строка, начинающаяся с буквы и состоящая из букв и цифр, знака доллара и знака подчеркивания), ***пробел*** – обязательный пробельный знак перед мнемоникой, ***мнемоника*** – мнемоническое обозначение команды микропроцессора, директивы ассемблера или имени макроопределения; ***операнд*** – целочисленная константа (двоичная вида 0110B, восьмеричная вида 3771Q, десятичная вида 356, шестнадцатеричная вида 0A9Dh или 0xFF12, литерал вида ‘A’), константа с плавающей запятой вида -0,123e-12; строка вида “abcd 12”, метка, временное имя вида $1234 или метка со знаком вопроса на конце вида M34_x?, константное выражение с операциями -, +, ~, !, *, /, %, <<, >>, <, <=, >, >=, =, ==, !=, &, ^, | и встроенными функциями; ***комментарий*** – пояснения к оператору.

1.2.2.1 Встроенные функции

Ассемблер поддерживает вычисление во время ассемблирования следующих числовых функций:

$acos(x) – арккосинус x в формате с плавающей запятой,

$asin(x) – арксинус x в формате с плавающей запятой,

$atan(x) – арктангенс x в формате с плавающей запятой,

$ceil(x) – наименьшее целое, не большее чем x,

$cos(x) – косинус x в формате с плавающей запятой,

$cosh(x) – гиперболический косинус x в формате с плавающей запятой,

$cvf(x) – преобразование x в формат с плавающей запятой,

$cvi(x) – преобразование x в целое число,

$exp(x) – натуральная экспонента x,

$fabs(x) – абсолютное значение x в формате с плавающей запятой,

$floor(x) – наибольшее целое, не большее x,

$fmod(x, y) – остаток от деления x на y,

$int(x) – возвращает 1 если x – целое, иначе – 0,

$ldexp(x, y) – возвращает x умноженное на 2 в степени y,

$log(x) – натуральный логарифм x,

$log10(x) – десятичный логарифм x,

$max(x, y) – максимальное из чисел x и y,

$min(x, y) – минимальное из чисел x и y,

$pow(x, y) – возвращает 2 в степени y,

$round(x) – округление до ближайшего целого,

$sgn(x) – знак числа x,

$sin(x) – синус x в формате с плавающей запятой,

$sinh(x) – гиперболический синус x в формате с плавающей запятой,

$sqrt(x) – квадратный корень x в формате с плавающей запятой,

$strtod(a) – преобразует строку a в число в формате с плавающей запятой,

$tan(x) – тангенс x в формате с плавающей запятой,

$tanh(x) – гиперболический тангенс x в формате с плавающей запятой,

$trunc(x) – округление до целого в направлении к нулю,

а также строковых функций:

$symlen(a) – длина строки a,

$symcmp(a ,b) – -1 – если a меньше b, 0 – если строки равны, и 1 – если больше,

$firstch(a, c) – индекс первого вхождения знака c в строку a,

$lastch(a, c) – индекс последнего вхождения знака c в строку a,

$isdefed(a) – 1 – если a определено в таблице имен, иначе – 0,

$ismember(a, b) – 1 – если в списке b имеется строка a, иначе – 0,

$iscons(a) – 1 – если a двоичная константа, 2 – восьмеричная, 3 – шестнадцатеричная, 4 – строковая, 5 – десятичная,

$isname(a) – 1 – если строка a является именем, иначе – 0,

$isreg(a) – 1 – если строка a является именем регистра, иначе – 0.

1.2.2.2 Режимы ассемблирования

Для того чтобы ассемблер мог правильно порождать код, ему необходимо сообщить о режиме работы микропроцессора. Для этих целей используются следующие директивы:

- .arms_on – ассемблирование при флаге ARMS, равном 1;
- .arms_off – ассемблирование при флаге ARMS, равном 0;
- .c54cm_on – ассемблирование при флаге C54CM, равном 1;
- .c54cm_off – ассемблирование при флаге C54CM, равном 0;
- .cpl_on – ассемблирование при флаге CPL, равном 1;
- .cpl_off – ассемблирование при флаге CPL, равном 0;
- .sst_on– ассемблирование при флаге SST, равном 1;
- .sst_off – ассемблирование при флаге SST, равном 0;
- .dp x – ассемблирование при заданном значении x регистра DP.

1.2.2.3 Секционирование

Для задания секций с неинициализированными данными используются директивы .bss (неименованная секция) и .usect (именованная секция):

.bss ***имя***, ***размер***[, ***блок***[, ***выравнивание***]];

.usect ”***имя-секции***”, ***размер***[, ***блок***[, ***выравнивание***]],

где ***имя*** – идентификатор резервируемой области памяти, ***размер*** – объем резервируемой области памяти в словах, ***блок*** – флаг выделение памяти в пределах одной страницы памяти, ***выравнивание*** – флаг выравнивания по границе слова, ***имя-секции*** – идентификатор секции данных. Элементы синтаксиса директив, заключенные в квадратные скобки, являются необязательными и могут быть опущены.

Для задания секций с инициализированными данными используются директивы .text (секция кода), .data (неименованная секция данных) и .sect (именованная секция данных):

.text;

.data;

.sect "***имя-секции***".

1.2.2.4 Инициализация данных

Для инициализации данных используются следующие директивы:

.[u]byte ***значение***[, ***значение*** …]; 16-разрядный байт;

.[u]char ***значение***[, ***значение*** …]; 16-разрядный знак;

.cstring "***строка***"[,"***строка***" …]; строка в формате языка C и C++;

.double ***значение***[, ***значение*** …]; число с плавающей запятой 64 бита;

.field ***значение***[, ***размер***]; упакованное битовое поле заданного размера;

.float ***значение***[, ***значение*** …]; число с плавающей запятой 32 бита;

.[u]half ***значение***[, ***значение*** …]; 16-разрядное целое число;

.[u]int ***значение***[, ***значение*** …]; 16-разрядное целое число;

.ivec ***адрес***[, ***стек***] ; 32-разрядный вектор прерывания;

.ldouble ***значение***[, ***значение*** …]; число с плавающей запятой 64 бита;

.[u]long ***значение***[, ***значение*** …]; 32-разрядное целое число;

.pstring "***строка***"[, "***строка***" …]; строка с упакованными в байты знаками;

.[u]short ***значение***[, ***значение*** …]; 16-разрядное целое число;

.string "***строка***"[, "***строка***" …]; строка без управляющих знаков;

.[u]word ***значение***[, ***значение*** …]; 16-разрядное целое число;

.[x]float ***значение***[, ***значение*** …]; число с плавающей запятой 32 бита;

.[x]long ***значение***[, ***значение*** …]; 32-разрядное целое число,

где u – префикс беззнакового формата чисел; x – префикс размещения данных без выравнивания; ***значение*** – константа, выражающая значение байта, знака, числа с

плавающей запятой и целого числа соответствующей разрядности (см. таблицу 1.7); ***строка*** – последовательность знаков в формате языка С (.cstring), в упакованном формате (.pstring) и в формате без управляющих последовательностей знаков языка С (.string); ***размер*** – число разрядов битового поля; ***адрес*** – метка процедуры обработки прерываний; ***стек*** – режим работы стека (NO_RETA – медленный вызов прерываний и возврат из прерываний без использования регистров RETA и CFCT, или USE_RETA – быстрый вызов и возврат благодаря использованию регистров RETA и CFCT).

1.2.2.5 Выравнивание

Для выравнивания данных в памяти используются следующие директивы:

.align [***размер***];

.even;

.space ***размер***;

.localalign,

где ***размер*** – в директиве .align число, которому должен быть кратен адрес следующего элемента данных текущей секции, в директиве .space – объем резервируемой памяти в словах, директива .even эквивалентна директиве .align с границей, равной 2, а директива .localalign выравнивает команды, следующие за ней так, чтобы они загружались по границе очереди команд.

1.2.2.6 Перекрестные ссылки

Для объявления перекрестных ссылок используются директивы:

.def ***имя*** [, ***имя*** …]];

.ref ***имя*** [, ***имя*** …]];

.global ***имя*** [, ***имя*** …]].

Директива .def используется для объявления одного или нескольких имен внутри текущего модуля, которые должные быть доступны в других модулях программы. Директива .ref используется для объявления одного или нескольких имен, используемых в текущем модуле, но определенных в других модулях. Директива .global объявляет имя глобальным и эквивалентна директиве .def или .ref в зависимости от места объявления имени.

1.2.2.7 Локальные имена

Для объявления локальных имен, т.е. имен, определенных в текущем модуле и недоступных в других модулях, используются директивы:

.asg ["]***строка***["], ***имя***; присваивание имени строки;

имя .equ ***значение***; присваивание имени значения;

имя .set ***значение***; присваивание имени значения (эквивалентно .equ);

имя .eval ***выражение***; присваивание имени значения выражения;

.label ***имя***; определение имени-метки в секции кода или данных;

.newblock; очистка всех локальных имен;

.unasg ***имя*** - удаление имени;

.undefine ***имя***; удаление имени (эквивалентна .unasg);

.var ***имя***[, ***имя*** …]; объявление имен внутри макроопределения.

1.2.2.8 Условное ассемблирование

В ассемблере реализованы две директивы условного ассемблирования:

– директива условного блока;

– директива блока повторения.

Директива условного блока:

```
.if выражение          ; начало условного блока;
…                      ; операторы, выполняющиеся при истинности выражения 1;
[.elseif выражение ]   ; начало условного блока
…                      ; операторы, выполняющиеся при истинности выражения 2;
[.else ]               ; начало безусловного блока;
…                      ; операторы, выполняющиеся в противном случае;
.endif                 ;  конец условного блока.
```

Директива блока повторения:

```
.loop [выражение]      ; начало блока [с условием повторения];
…                      ; операторы тела блока повторения;
[.break [выражение]] ; выход из блока [при истинном значении выражения];
…                      ; операторы тела блока повторения;
.endloop               ; конец блока повторения.
```

1.2.3 Компоновщик

В процессе компоновки объектные файлы объединяются по секциям (рисунок 1.37). В каждом объектом файле присутствует описание и определено содержимое следующих секций:

– кода .text;

– инициализируемых данных .data;

– не инициализируемых данных .bss;

– других именованных секций (Init и Tables на рисунке 1.37).

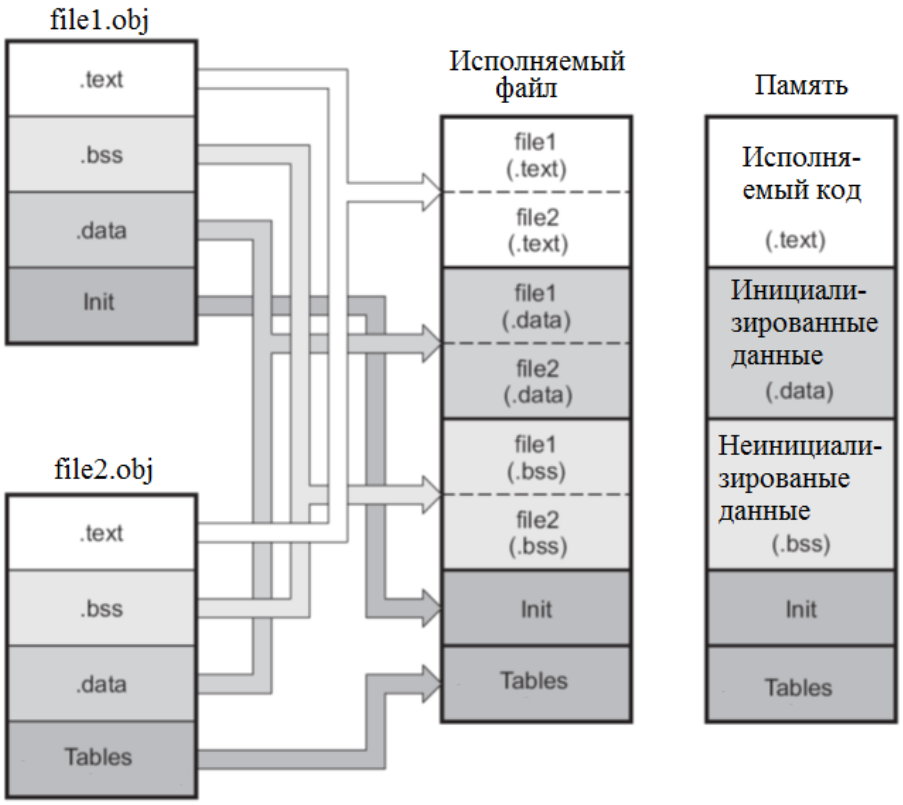


Рисунок 1.37 – Компоновка и загрузка

1.2.3.1 Командный файл

Для описания процессов компоновки и загрузки исполняемого файла создается командный файл компоновщика (см. приложение Е). В командном файле могут быть заданы следующие команды, описывающие компоновку исполняемого файла и его загрузку в физическую память [43, с. 228]:

-stack ***размер*** – размер секции .stack для основного стека;

-sysstack ***размер*** – размер секции .sysstack для вспомогательного стека;

-heap ***размер*** – размер секции .system для динамически выделяемой памяти;

-e ***имя*** – задание имени точки входа в программу, по умолчанию _main;

-u ***имя*** – удаление имени из таблицы не найденных ссылок;

-c – флаг инициализации переменных во время загрузки программы;

директива описания сегментов памяти MEMORY;

директива описания секций программы SECTIONS.

1.2.3.2 Сегментация памяти

Для описания адресных пространств и сегментов физической памяти используется директива MEMORY:

MEMORY {

[PAGE ***номер***:] ***имя*** [(***атрибуты***)]: origin=***адрес***, length=***длина***[, fill=***константа***]

…

},

где ***номер*** – идентификатор независимого адресного пространства памяти; ***имя*** – имя области памяти, к которой в директиве SECTIONS привязываются одна или несколько секций программы; ***атрибуты*** – один или несколько флагов доступа (R – разрешено чтение, W – разрешена запись, X – разрешено выполнение кода, I – разрешена инициализация данных); ***адрес*** (***длина***) – начальный адрес байта (объем в байтах) области памяти в соответствующем адресном пространстве; ***константа*** – двухбайтовое целое число, используемое для заполнения тех участков области памяти, которые на распределены для размещения секций программы.

1.2.3.3 Секционирование памяти

Директива секционирования SECTIONS определяет отображение секции кода и данных в сегменты физической памяти:

SECTIONS {

имя: [***свойство*** [, ***свойство***] …]]

…

},

где для каждой секции с идентификатором ***имя*** могут быть заданы одно или несколько ***свойств***, регламентирующих размещение секции, ее тип и список привязываемых объектных файлов.

Для задания размещения секции используется одна из форм выражения свойства load:

load = ***сегмент***;

> ***сегмент***;

>> ***сегмент*** | ***сегмент*** | ***сегмент*** … ;

адрес,

где ***сегмент*** – имя сегмента физической памяти (имя адресного пространства), описанного в директиве MEMORY, ***адрес*** – прямое задание начального адреса загрузки секции.

Тип секции задается с помощью свойства type,

type = ***тип***,

где ***тип*** может принимать следующие значения:

– DSECT – муляжная секция, которая не включается в исполняемый файл и не загружается в физическую память;

– COPY – фиктивная секция, отличающаяся от секции DSECT тем, что включается в исполняемый файл, но не загружается в физическую память;

– NOLOAD – секция, не загружаемая в физическую память, но память для которой все же выделяется в процессе компоновки.

Список объектных файлов и их секций, привязываемых к описываемой в директиве секции, задается в фигурных скобках,

{ ***файл*** (***секция***) ***файл*** (***секция***) … }

где ***файл*** – имя или шаблон имени объектного файла, ***секция*** – имя привязываемой секции файла.

1.2.3.4 Предопределенные имена

В процессе компоновки исполняемого файла компоновщик создает имена, доступные в качестве внешних (extern) идентификаторов в программах на языке С и С++:

__STACK_END – адрес байта, следующего за секцией .stack;

__STACK_SIZE – размер в байтах секции .stack.

_SYSSTACK_SIZE – размер в байтах секции .sysstack;

__SYSMEM_SIZE – размер в байтах секции .sysmem.

Помимо этого, создаются имена, доступные не только в программах на языке С и С++, но и в программах на языке ассемблера (объявляются директивой .global):

.text – адрес первого байта секции кода .text;

etext – адрес байта, следующего за секцией .text;

.data – адрес первого байта секции инициализируемых данных .data;

edata – адрес байта, следующего за секцией .data;

.bss – адрес первого байта секции не инициализируемых данных .bss;

end – адрес байта, следующего за секцией .bss.

1.3 Домашнее задание 1

1.3.1 Общие указания

Целью выполнения домашнего задания является изучение команд микропроцессора, заданных в теме домашнего задания. Темы домашних заданий приведены в пп. 1.3.2.

Для выполнения домашнего задания предварительно необходимо ознакомиться с технической документацией [45], найти и изучить описание команд микропроцессора, указанных в теме домашнего задания.

Следующим этапом выполнения домашнего задания является установление списка исследуемых команд, описание форматов команд и способов адресации их операндов. Если команд окажется несколько, то выбрать из найденных команд две или три характерные.

На последнем этапе выполнения домашнего задания разрабатывается методика исследования команд, позволяющая в лабораторных условиях установить те особенности выполнения команд, которые не были ясны из имеющихся описаний. Для проверки выполнения исследуемых команд подготавливаются тестовые примеры, состоящие исходных и результирующих данных исследуемых команд.

Итогом выполнения домашнего задания является оформление отчета. Срок выполнения домашнего задания – две недели с момента выдачи темы индивидуального задания.

1.3.2 Индивидуальные задания

Ниже приведены темы индивидуальных заданий, заключающиеся в исследовании следующих команд микропроцессора:

1) адресной арифметики AADD, ASUB [45, с. 5-2, 5-85];
2) вычисления абсолютного расстояния ABDST [45, с. 5-9];
3) двойного сложения ADD [45, с. 5-35];
4) сложения с параллельным сохранением ADD:MOV [45, с. 5-40];
5) двойного сложения и вычитания ADDSUB, SUBADD [45, с. 5-42, 5-626];
6) сложения с абсолютным значением ADDV [45, с. 5-54];
7) модификации адресных регистров AMAR [45, с. 5-56];
8) модификации регистра с умножением и сложением AMAR::MAC [45, с. 5-60];

9) адресной пересылки AMOV [45, с. 5-69];
10) безусловного перехода B, BAND [45, с. 5-91];
11) условного перехода BCC, BCCU [45, с. 5-96];
12) подсчета и модификации бит BCNT, BNOT, BCLR, BSET [45, с. 5-111];
13) расширения и извлечения битовых полей BFXPA, BFXTR [45, с. 5-112];
14) проверки бит BTST, BTSTCLR, BTSTNOT, BTSTSET, BTSTP [45, с. 5-121];
15) вызова и возврата из подпрограмм CALL, RET [45, с. 5-131, 5-518];
16) условного вызова и возврата CALLCC, RETCC [45, с. 5-135, 5-520];
17) сравнения с логическими операциями CMPAND, CMPOR [45, с. 5-145];
18) задержки DELAY [45, с. 5-156];
19) симметричной нерекурсивной фильтрации FIRSADD [45, с. 5-158];
20) кососимметричной нерекурсивной фильтрации FIRSSUB [45, с. 5-160];
21) ожидания сброса или прерывания IDLE [45, с. 5-162];
22) прерывания и возврата из прерывания INTR, RETI [45, с. 5-163, 5-522];
23) квалификатора чтения-модификации-записи .LK [45, с. 5-165];
24) вычисления наименьших средних квадратов LMS, LMSF [45, с. 5-167];
25) квалификаторов линейной и циклической адресации .LR и .CR [45, с. 5-173];
26) умножения со сложением и параллельной задержкой MACMZ [45, с. 5-191];
27) двух параллельных умножений со сложением MAC::MAC [45, с. 5-193];
28) умножения со сложением и вычитанием MAC::MAS [45, с. 5-228];
29) умножения со сложением и умножением MAC::MPY [45, с. 5-248];
30) умножения со сложением и загрузкой из памяти MACM::MOV [45, с. 5-267];
31) нормализации MANT::NEXP [45, с. 5-272];
32) умножений с вычитанием MAS::MAS [45, с. 5-297];
33) умножения с вычитанием и умножением MAS::MPY [45, с. 5-309];
34) умножения с вычитанием и загрузкой из памяти MASM::MOV [45, с. 5-318];
35) минимума и максимума MIN, MAX [45, с. 5-322, 5-331];
36) минимума и максимума MINDIFF, MAXDIFF [45, с. 5-325, 5-334];
37) квалификатора доступа к регистрам MMAP [45, с. 5-340];
38) модификаторов операндов в команде умножения MPY [45, с. 5-430];
39) преобразований операндов в командах пересылки MOV [45, с. 5-342];
40) умножения и умножения с вычитанием MPY::MAS [45, с. 5-458];
41) двойного умножения MPY::MPY [45, с. 5-468];
42) доступа к стеку POP, PSH [45, с. 5-496, 5-506];

43) доступа к стеку POPBOTH, PSHBOTH [45, с. 5-503, 5-513];
44) квалификатора ввода-вывода PORT [45, с. 5-504];
45) циклических сдвигов ROL, ROR [45, с. 5-524];
46) логических и арифметических сдвигов SFTL, SFTS [45, с. 5-559, 5-565];
47) округления ROUND [45, с. 5-528];
48) простого повторения RPT [45, с. 5-530];
49) простого повторения RPTADD, RPTSUB [45, с. 5-535, 5-553];
50) блочного повторения RPTB [45, с. 5-538];
51) насыщения SAT [45, с. 5-555];
52) условных сдвигов SFTCC [45, с. 5-557];
53) возведения в квадрат SQA, SQS, SQR, SQDST [45, с. 5-579];
54) условного вычитания SUBC [45, с. 5-631];
55) перестановки SWAP, SWAPP, SWAP4 [45, с. 5-634];.
56) отладочного прерывания TRAP [45, с. 5-646];
57) условного выполнения в адресной фазе конвейера XCC [45, с. 5-648];
58) условного выполнения в фазе исполнения конвейера XCCPART [45, с. 5-648].

1.3.3 Пример выполнения домашнего задания

Оформление отчета по домашнему заданию осуществляется в соответствии с ГОСТ 2.105-95 [8], список литературы – по ГОСТ 7.1-2003 [9]. Отчет должен содержать:

- титульный лист (приложение В);
- содержание;
- постановка задачи;
- описание команд;
- форматы команд;
- методика исследования;
- список использованной литературы.

Ниже приведен пример выполнения и оформления домашнего задания на тему «Исследование поразрядных логических команд микропроцессора».

1.3.3.1 Постановка задачи

Темой домашнего задания является исследование команд микропроцессора TMS230C5515, реализующих поразрядные логические операции. Для выполнения домашнего задания необходимо:

– ознакомиться с рекомендованной литературой;

– изучить форматы команд и способов адресации операндов;

– разработать методику исследования команд;

– подготовить тестовые данные для проверки выполнения команд;

– оформить отчет по домашнему заданию.

Основным результатом домашнего задания является методика лабораторного исследования поразрядных логических команд, позволяющая путем выполнения проверочной последовательности команд микропроцессора проверить результаты обработки данных заданными командами и выявить особенности этой обработки.

1.3.3.2 Команды поразрядных логических операций

Из анализа источника [45] следует, что у микропроцессоров серии TMS320C55x реализованы следующие поразрядные логические команды:

– отрицание NOT;

– конъюнкция AND;

– дизъюнкция OR;

– неэквиваленция XOR.

1.3.3.2.1 Команда NOT

Команда NOT имеет следующий синтаксис:

NOT [src,] dst, (1.1)

где NOT – мнемоника команды отрицания, src – необязательный регистр-источник (AC0-AC3, AR0-AR7, T0-T3), dst – обязательный регистр-приемник (AC0-AC3, AR0-AR7, T0-T3).

Команда NOT выполняет изменение разрядов регистра-источника src на противоположное значение и записывает получившийся при этом результат в регистр-приемник dst. Если регистр-источник src в (1.1) не указан, то источником исходных данных команды является регистр-приемник dst.

В случае если регистр-приемник является аккумулятором AC0-AC3, то операция выполняется арифметико-логическим устройство операционного блока. По этой причине если регистр-источник дополнительный регистр AR0-AR7 или временный регистр T0-T3, то перед выполнением операции старшие разряды аккумулятора заполняются нулями.

В случае если регистр-приемник является дополнительным регистром AR0-AR7 или временным регистром T0-T3, то операция выполняется арифметико-логическим устройство блока адресации. По этой причине если в качестве регистра-источника

используется аккумулятор AC0-AC3, то операция выполняется только над младшими 16 разрядами этого регистра.

После выполнения команды NOT флаги состояния (статуса) микропроцессора не изменяются.

1.3.3.2.2 Команда AND

Команда конъюнкции AND имеет следующий синтаксис:

AND src, dst; (1.2)

AND k8, src, dst; (1.3)

AND k16, src, dst; (1.4)

AND Smem, src, dst; (1.5)

AND ACx << #SHIFTW[, ACy]; (1.6)

AND k16 << #16, [ACx,] ACy; (1.7)

AND k16 << #SHFT, [ACx,] ACy; (1.8)

AND k16, Smem, (1.9)

где src (dst) – регистр-источник (регистр-приемник) AC0-AC3, AR0-AR7, T0-T3; k8 (k16) – 8-разрядная (16-разрядная) операнд-константа, размещаемая в коде команды; Smem – 16-разрядный операнд в памяти; ACx, ACy – аккумулятор AC0-AC3; SHFT (SHIFTW) – 4-разрядное (6-разрядное) число сдвигов от 0 до 15 (от −32 до +31) 16-разрядного слова (32-разрядного двойного слова), выполняемое перед логической операцией и кодируемое непосредственно в команде.

Команда AND осуществляет поразрядное логическое умножение разрядов операндов-источников и записывает получившийся при этом результат в операнд-приемник dst. Команду AND выполняет операционный блок, если приемником результата является аккумулятор, или блок адресации – если приемником результата является дополнительный или временный регистр, а также 16-разрядная ячейка памяти.

Команда AND дополняет нулями 16-разрядные операнды до разрядности аккумулятора-приемника и отсекает старшие разряды аккумулятора-источника до разрядности 16-разрядного операнда-приемника.

Команда (1.2) ничем не отличает от ранее описанной команды NOT. В командах (1.3), (1.4), (1.7)-(1.9) в качестве одного из операндов-источников используются константы k8 или k16, а в команде (1.5) – адресуемая 16-разрядная ячейка памяти Smem. Команды (1.6)-(1.8) перед операцией конъюнкции сдвигают первый операнд на SHIFTW, 16 и SHFT разрядов. В свою очередь команда (1.9) выполняет конъюнкцию

константы k16 с адресуемой ячейкой памяти Smem, и в эту же ячейку записывает результат.

Как и для команды NOT после выполнения команды AND флаги результата не изменяются.

1.3.3.2.1 Команды OR и XOR

Команды OR и XOR имеют синтаксис, аналогичный синтаксису команды AND (1.2)-(1.9). Однако в отличие от команды AND, команда OR выполняет поразрядную логическую операцию ИЛИ, а команда XOR – поразрядную неэквиваленцию.

В связи с похожим синтаксисом и одинаковым форматом команд AND, OR и XOR для дальнейших исследований выберем команды NOT и AND.

1.3.3.3 Форматы команды и способы адресации операндов

1.3.3.3.1 Поля форматов команда

Поле адресации аккумулятора-источника SS и аккумулятора-приемника DD:

00 – аккумулятор AC0,

01 – аккумулятор AC1,

10 – аккумулятор AC2,

11 – аккумулятор AC3.

Поле адресации регистров-источников FSSS и регистров приемников FDDD:

0000 – аккумулятор AC0,

0001 – аккумулятор AC1,

0010 – аккумулятор AC2,

0011 – аккумулятор AC3,

0100 – временный регистр T0,

0101 – временный регистр T1,

0110 – временный регистр T2,

0111 – временный регистр T3,

1000 – дополнительный регистр AR0,

1001 – дополнительный регистр AR1,

1010 – дополнительный регистр AR2,

1011 – дополнительный регистр AR3,

1100 – дополнительный регистр AR4,

1101 – дополнительный регистр AR5,

1110 – дополнительный регистр AR6,

1111 – дополнительный регистр AR7.

Поле kk…k для хранения беззнаковых констант k8 и k16:

00…00 – константа 0,

00…01 – константа 1, …,

11…11 – константа 0FFh или 0FFFFh.

Поле адресации операнда Smem в памяти AAAA AAAI:

AAAA AAA0 – 7 разрядов смещения при прямой адресации операнда относительно регистров XDP или XSP в зависимости от флага CPL регистра статуса микропроцессора ST1,

AAAA AAA1 – 7 разрядов кода способа косвенной адресации операнда через регистры CDP, AR0-AR7 и T0-T1.

Поле способа косвенной адресации операнда в памяти AAAA AAA1:

0001 0001 – ABS16(#k16),

0011 0001 – *(#k23),

0101 0001 – port(#k16),

0111 0001 – *CDP,

1001 0001 – *CDP+,

1011 0001 – *CDP−,

1101 0001 – *CDP(#K16),

1111 0001 – *+CDP(#K16),

PPP0 0001 – *ARn,

PPP0 0011 – *ARn+,

PPP0 0101 – *ARn−,

PPP0 0111 – *(ARn + T0),

PPP0 1001 – *(ARn – T0),

PPP0 1011 – *ARn(T0),

PPP0 1101 – *ARn(#K16),

PPP0 1111 – *+ARn(#K16),

PPP1 0011 – *(ARn + T1) при ARMS = 0 и *ARn(#1) при ARMS = 1,

PPP1 0101 – *(ARn – T1) при ARMS = 0 и *ARn(#2) при ARMS = 1,

PPP1 0111 – *ARn(T1) при ARMS = 0 и *ARn(#3) при ARMS = 1,

PPP1 1001 – *+ARn при ARMS = 0 и *ARn(#4) при ARMS = 1,

PPP1 1011 – *−ARn при ARMS = 0 и *ARn(#5) при ARMS = 1,

PPP1 1101 – *(ARn + T0B) при ARMS = 0 и *ARn(#6) при ARMS = 1,

PPP1 1111 – *(ARn – T0B) при ARMS = 0 и *ARn(#7) при ARMS = 1,

где PPP – поле номера n дополнительного регистра ARn; k16 (k23, K16) – беззнаковая 16-разрядная (беззнаковая 23-разрядная, знаковая 16-разрядная) константа, хранящаяся в памяти непосредственно после команды; port – ключевое слово для обращения в адресное пространство ввода-вывода; ARMS – флаг режима косвенной адресации микропроцессора (3 разряд регистра статуса ST2); B – признак бит-реверсивной адресации.

1.3.3.3.2 Формат команды NOT

Команда отрицания NOT (1.1) имеет следующий формат:

0011 011E FSSS FDDD, (1.10)

где E – бит параллельности, FSSS (FDDD) – поле адресации регистра-источника (регистра-приемника).

Примеры кодирования команды NOT (1.10):

0011 0110 0001 0001 – NOT AC1,

0011 0110 0011 1001 – NOT AC3, AR1,

0011 0111 0101 0110 – || NOT T1, T2,

где || – признак параллельного выполнения команды, вызывающий установку поля E команды в единицу.

1.3.3.3.3 Формат команд AND

Команды конъюнкции AND (1.2)-(1.9) имеют следующие форматы соответственно:

0010 100E FSSS FDDD, (1.11)

0001 100E kkkk kkkk FDDD FSSS, (1.12)

0111 1101 kkkk kkkk kkkk kkkk FDDD FSSS, (1.13)

1101 1001 AAAA AAAI FDDD FSSS, (1.14)

0001 000E DDSS 0000 xxSH IFTW, (1.15)

0111 1010 kkkk kkkk kkkk kkkk SSDD 010x, (1.16)

0111 0010 kkkk kkkk kkkk kkkk SSDD SHFT, (1.17)

1111 0100 AAAA AAAI kkkk kkkk kkkk kkkk, (1.18)

где E – бит параллельности, FSSS (FDDD) – поле адресации регистра-источника (регистра-приемника), kk…k – поле непосредственных констант, AAAA AAAI – поле адресации операнда в памяти Smem, SS (DD) – поле адресации аккумулятора-источника SS (аккумулятора-приемника), SHIFTW (SHFT) – поле для числа сдвигов 32-разрядного (16-разрядного) операнда-источника, x – разряд формата команды, зарезервированный для дальнейшего расширения.

Примеры кодирования команд AND (1.11)-(1.18):

0010 1000 0000 0001 – AND AC0, AC1,

0001 1000 0101 1010 0011 1001 – AND #5Ah, AC3, AR1,

0111 1101 1111 1111 0000 0001 0100 01001 – AND #FF01h, T0, T1,

1101 1001 0010 0011 0000 1010 – AND *AR1+, AC0, AR2

0001 0000 0011 0000 0010 1111 – || AND AC1<<#-17, AC0

0111 1010 1010 1011 1100 1101 0101 0100 – AND #ABCDh<<#16, AC1

0111 0010 0000 0100 1101 0010 0100 1011 – AND #1234<<#11, AC1, AC0,

1111 0100 1010 1111 1010 1010 0101 0101 – AND #AA55h, *+AR5(#12),

где || – признак параллельного выполнения команды.

1.3.3.4 Методика исследования

Методика исследования команд разрабатывается для выявления тех особенностей выполнения команд, которые не ясны из имеющихся описаний.

1.3.3.4.1 Исследование команды NOT

Исследуем поведение команды NOT при расширении регистра-источника до разрядности аккумулятора и усечение аккумулятора до разрядности регистра приемника при различной разрядности арифметико-логического устройства операционного блока.

Шаг 1. Переведем микропроцессор в 32-разрядный режим работы арифметико-логического устройства операционного блока. Для этого запишем в память программ команду очистки флага M40 в регистре статуса ST1:

0100 0110 1010 0010 – BCLR 10, ST1,

где 10 – номер бита флага M40 в регистре статуса ST1.

Шаг 2. Закодируем и запишем в память программ следующие две команды NOT:

0011 0110 0000 1000 – NOT AC0, AR0;

0011 0110 1001 0001 – NOT AR1, AC1.

Шаг 3. Занесем в регистры микропроцессора исходные данные:

AC0 – F012481234h,

AR0 – 1503h,

AC1 – 0F12480F0Fh,

AR1 – 4321h.

Шаг 4. Выполним команды из шагов 1, 2 и занесем в протокол исследования содержимое регистров из шага 3. Ожидаемый результат выполнения команд:

AC0 – F012481234h,

AR0 – ~ F012481234 = EDCBh,

AC1 – ~ 00004321h = 0FFFFFBCDEh,

AR1 – 4321h,

где ~ – знак операции отрицания.

Шаг 5. Переведем микропроцессор в 40-разрядный режим работы арифметико-логического устройства операционного блока командой BSET:

0100 0110 1010 0011 – BSET 10, ST1.

Шаг 6. Повторим шаги 2 – 4. Ожидаемый результат выполнения команд:

AC0 – F012481234h,

AR0 – ~ F012481234 = EDCBh,

AC1 – ~ 0000004321h = FFFFFFBCDEh,

AR1 – 4321h.

Шаг 7. Проанализируем полученные данные и сделаем выводы.

1.3.3.4.2 Исследование команды AND

Исследуем команду AND (1.6) в 32-разрядном и 40-разрядном режимах работы арифметико-логического устройства операционного блока при числе сдвигов, равном минус 32.

Шаг 1. Переведем микропроцессор в 32-разрядный режим работы путем очистки флага M40 в регистре статуса ST1:

0100 0110 1010 0010 – BCLR 10, ST1.

Шаг 2. Закодируем и запишем в память программ исследуемую команду AND:

0001 0000 0100 0000 0011 1111 – AND AC0<<#-32, AC1.

Шаг 3. Занесем в регистры микропроцессора исходные данные для команды:

AC0 – 0123456789h,

AC1 – 0F0F0F0F0Fh.

Шаг 4. Выполним команды из шагов 1, 2 и запишем в протокол исследования содержимое регистров из шага 3. Ожидаемый результат:

AC0 – 0123456789h,

AC1 – 0123456789h <<-32 & 0F0F0F0F0Fh = 0000000000h.

Шаг 5. Переведем микропроцессор в 40-разрядный режим работы командой BSET:

0100 0110 1010 0011 – BSET 10, ST1.

Шаг 6. Повторим шаги 2 – 4. Ожидаемый результат:

AC0 – 0123456789h,

AC1 – 0123456789h <<-32 & 0F0F0F0F0Fh = 0000000001h.

Шаг 7. Проанализируем полученные данные и сделаем выводы.

1.4 Лабораторная работа 1

1.4.1 Общие указания

Целью лабораторной работы является знакомство с интегрированной средой разработки программ Code Composer Studio версии 5 (CCS) компании Texas Instruments. Среда CCS предназначена для разработки программного обеспечения для микропроцессоров и микроконтроллеров, выпускаемых компанией TI, и включает такие инструменты, как редактор исходных текстов, компилятор, компоновщик, отладчик и симуляторы [68].

Лабораторная работа заключается в модификации в соответствие с выданным индивидуальным заданием текстов несложных программ на языке С (Приложение А) и ассемблера (Приложение Б), создании нового проекта в CCS, компиляции и отладке модифицированных программ с помощью симулятора микропроцессора TMS320C5515.

Для подготовки к лабораторной работе 1 необходимо повторить лекционный материал по архитектуре и организации микропроцессора, изучить исходные тексты программ из приложений С-Е и разработать модули на языке С и на языке ассемблера, реализующие программу исследований из домашнего задания 1.

1.4.2 Выполнение лабораторной работы

1.4.2.1 Создание проекта

1.4.2.1.1 Для создания проекта выберите пункты меню *File* → *New* → *CCS Project*. Откроется форма нового проекта.

1.4.2.1.2 В поле *Project Name* введите имя проекта, например, *MPUOS_LAB1*. В ниспадающем списке *Output Type* выберите *Executable* для создания исполняемой программы.

1.4.2.1.3 При установке флага *Use default location* проект будет создан в папке текущего пользователя. В противном случае место размещения папки проекта следует выбрать после нажатия на кнопку *Browse*.

1.4.2.1.4 В разделе *Device* в ниспадающем списке *Famaly* выберите семейство микропроцессоров *C5500*, а в списках *Variant* тип и модель микропроцессора – *C551x* и *TMS320C5515*. Если в проекте используется симулятор, то поле *Connection* оставьте пустым. Если в проекте используется внешняя оценочная плата TMS320C5515 EVM, выберите эмулятор *Spectrum Digital DSK-EVM-eZdsp on board USB emulator*.

1.4.2.1.5 В случае необходимости в разделе *Advanced Setting* выберите в поле *Device endianness* порядок слов в многословных числах *big* (число записывается в память в порядке увеличения адресов начиная со старшего слова и заканчивая младшим), в поле *Compile version* – версию компилятора *TI v4.4.1*, в поле *Output format* – формат исполняемого файла *Legacy COFF*, в поле *Linked command file* – командный файл компоновщика, а в поле *Runtime support library* – стандартную библиотеку времени исполнения, например, *rts55.lib*.

1.4.2.1.6 В разделе *Project Template and Examples* выберите пустой проект *Empty Project (with main.c)*. Нажмите кнопку *Finish*. CCS создаст проект, который отобразится в окне *Project Explorer*.

1.4.2.1.7 Для включения в проект новых файлов используйте контекстное меню, открывающееся при нажатии правой кнопки мыши на имени проекта в панели *Project Explorer*. В этом меню выберите пункт *New* и *File* и введите имя файла в поле *File name*, например, *test.asm*, и нажмите кнопку *Finish*.

1.4.2.1.8 Для добавления в проект существующих файлов в панели *Project Explorer* установите указатель на проект, выберите в оконном меню пункты *Project* → *Add Files* и в появившемся окне *New Source File* задайте место размещения добавляемого файла. При установленном флаге *Copy File* выбранный файл будет скопирован в папку проекта.

1.4.2.1.9 Введите подготовленные исходные тексты программ в окна текстового редактора *main.c* и *test.asm*. В окно *C5515.cmd* скопируйте командный файл компоновщика из Приложения Е.

1.4.2.2 Сборка проекта

1.4.2.2.1 Выполните сборку проекта, выбрав пункт меню *Project* → *Build Project*. Просмотрите в окне *Console* описания ошибок, обнаруженных компилятором, ассемблером и компоновщиком. Исправьте обнаруженные ошибки и повторите сборку проекта.

1.4.2.2.2 В случае необходимости измените параметры сборки, выбрав в контекстном меню проекта (окно *Project Explorer*) пункт *Build Options*. Аналогичные

настройки можно выполнить для каждого файла проекта через контекстное меню, открывающееся в окне *Project Explorer* при нажатии правой кнопки мыши на имени файла.

1.4.2.3 Выбор целевой платформы

1.4.2.3.1 До отладки проекта необходимо выбрать и сконфигурировать целевую платформу. Целевой платформой может быть либо симулятор, либо эмулятор. В последнем случае к компьютеру должна быть подключена оценочная плата. Для выбора целевой платформы создайте конфигурационный файл, для чего в контекстном меню проекта выберите пункт *New* → *Target Configuration File*.

1.4.2.3.2 В открывшемся окне задайте имя конфигурационного файла, например, *C5515_simulator*, к которому будет добавлено стандартное расширение *.ccxml*. По завершению ввода нажмите кнопку *Finish*.

1.4.2.3.3 В появившейся вкладке с именем созданного конфигурационного файла (*C5515_simulator.ccxml*) выберите в ниспадающем списке *Connection* раздел симуляторов *Texas Instruments Simulator*, и появившемся списке симуляторов установите флажок напротив симулятора *C55x CPU Functional Simulator* или *C55x CPU Cycle Accurate Simulator* последней версии.

1.4.2.3.4 Завершите конфигурирование, нажав на кнопку *Save* в окне конфигурационного файла. Проконтролируйте, что конфигурационный файл появился в списке файлов проекта с установленным атрибутом активности *Active/Default*.

1.4.2.4 Отладка программы

1.4.2.4.1 Установите фокус ввода на имени проекта в окне *Project Explorer* и выберите пункт меню *Run* → *Debug*. По этой команде запуститься отладчик и созданные ранее исполняемый файл с именем проекта и расширением *.out* загрузится для исполнения в память целевой платформы.

1.4.2.4.2 Для выполнения загруженной программы по шагам используйте пункты меню *Run* → *Step Into (F5)* или *Run* → *Step Over (F6)*, а для запуска программы – *Run* → *Resume (F8)*.

1.4.2.4.3 При пошаговой отладке программы содержимое регистров микропроцессора контролируйте в окне *Registers*, содержимое ячеек памяти – в окне *Memory*, текст программы на языке С с подсвеченным текущим оператором – в окне редактора соответствующего исходного файла, а исполняемый код программы на языке ассемблера – в окне *Disassembly*.

1.4.2.4.4 Для ускорения прохода отладчика до требуемого оператора (команды) используйте точки останова. Для задания точки останова установите фокус ввода на требуемый оператор (команду) и выберите в контекстном меню пункт *New Breakpoint* или *Toggle Breakpoint*. После установки точек останова используйте команду запуска программы *Target → Run (F8)*, которая приведет к выполнению программы до тех пор, пока не наступит очередь выполнения команды, помеченной точкой останова.

1.4.2.4.5 Для завершения отладки выберите в оконном меню пункт *Run → Terminate (Ctrl-F2)*.

1.4.2.5 Выполнение индивидуального задания

1.4.2.5.1 Используя отладчик, выполните трассировку программы исследований команд микропроцессора, заданных в индивидуальном задании.

1.4.2.5.2 При трассировке фиксируйте в протоколе исследований содержимое регистров процессора и используемых ячеек памяти.

1.4.2.5.3 В случае необходимости остановите отладку, внесите изменение в исходные тексты программ, перекомпилируйте проект и повторите процесс отладки.

1.4.3 Требования к отчету

Отчет по домашнему заданию должен содержать:

- титульный лист (приложение В);
- содержание;
- описание задания;
- результаты выполнения задания;
- выводы и рекомендации;
- список использованной литературы;
- приложения с текстами программ.

Оформление отчета осуществляется в соответствии с ГОСТ 2.105-95 [8], список литературы – по ГОСТ 7.1-2003 [9].

1.5 Контрольные вопросы 1

Контроль по модулю 1 заключается в обосновании разработанной для лабораторной работы 1 программы, предназначенной для исследования особенностей

выполнения команд микропроцессора, объяснении полученных экспериментальных данных и защите сделанных выводов.

Для оценки глубины усвоения учебного материала и проверки готовности к модульному контролю рекомендуется ответить на следующие контрольные вопросы.

Вопрос 1.1. Как связана тактовая частота микропроцессора с напряжением питания и мощностью потребления? Для чего предоставляется возможность в программе изменять тактовую частоту микропроцессора?

Вопрос 1.2. Для чего необходимо постоянное запоминающее устройство в составе микропроцессора? Зачем предоставлена возможность отключать постоянное запоминающее устройство после инициализации микропроцессора?

Вопрос 1.3. С какой целью внутри микропроцессора размещена одновходовая и двухвходовая основная память? Для решения какой проблемы используется двухвходовая память?

Вопрос 1.4. Для каких задач может потребоваться интерфейс внешней памяти микропроцессора? Почему внешняя память микропроцессора разделена на синхронную и асинхронную память?

Вопрос 1.5. Чем отличается шина от магистрали? Почему в составе микропроцессора имеется 6 комплектов шин и магистралей? По какой причине используются раздельные шины и магистрали для чтения и записи данных, а также для чтения команд и чтения-записи данных?

Вопрос 1.6. По какой причине микропроцессор имеет два адресных пространства: адресное пространство памяти и адресное пространство ввода-вывода?

Вопрос 1.7. Почему число разрядов в шине адреса для чтения программ на один больше, чем число разрядов в шинах адресов для чтения-записи данных? Почему программы и данные храниться в одном и том же адресном пространстве?

Вопрос 1.8. Почему 32-разрядные операнды операционного блока не могут размещаться во внешней памяти?

Вопрос 1.9. Какой объем данных может быть считан и записан одной командой микропроцессора?

Вопрос 1.10. Какие решения, доступные из программ, используются для понижения электропотребления микропроцессора?

Вопрос 1.11. Каков признак переполнения разрядной сетки для знакового и беззнакового, целочисленного и дробного представления чисел? Для решения каких задач используется флаг переноса за пределы разрядной сетки?

Вопрос 1.12. Почему только целочисленного формата представления чисел недостаточно для цифровой обработки сигналов?

Вопрос 1.13. Почему обработка сигналов, представленных числами в формате с плавающей запятой, менее эффективна, чем обработка сигналов, представленных числами в формате фиксированной запятой?

Вопрос 1.14. Как кодируются дробные и целые числа со знаком? Зависит ли операция изменения знака числа от формата его представления?

Вопрос 1.15. Чем отличаются форматы представления дробных чисел Q1.15, Q4.12, Q1.31 и Q9.31? Почему для представления дробных чисел используется столько форматов?

Вопрос 1.16. Какое число записано в ячейку памяти с содержимым 8105h в формате N16, Z16, Q1.15, Q4.12?

Вопрос 1.17. Представьте числа –0,989 и 0,357 в формате Q1.15, Q4.12 и Q1.31, а числа 5749 и –7475 – в формате N16 и Z16.

Вопрос 1.18. Чем вызвано использование при цифровой обработке сигналов порядка слов в многословных числах «от старшего к младшему»?

Вопрос 1.19. Преобразуйте 32-разрядные числа FFFFFF3Dh, 3000A9C3h и 00018423h, заданные в формате Q1.31, в 16-разрядные числа в формате Q1.15 с математическим и банковским округлением.

Вопрос 1.20. Преобразуйте 16-разрядные числа 80A4h, 70EBh и 000Fh, заданные в формате Q1.15, в 32-разрядные числа в формате Q1.31.

Вопрос 1.21. Преобразуйте 40-разрядные числа FFFFFF00EEh, FE3000A9C3h и 0100018423h, заданные в формате Q9.31, в 32-разрядные числа в формате Q1.31 с насыщением и без насыщения.

Вопрос 1.22. Преобразуйте 40-разрядные числа FFFFFF8000h, 000002803Eh, FE30008000h и 0100018000h, заданные в формате Q9.31, в 16-разрядные числа в формате Q1.15 с насыщением и без насыщения, с математическим и банковским округлением.

Вопрос 1.23. Почему разрядность регистров-аккумуляторов равна 40 разрядам и больше разрядности данных, хранимых в памяти? Для чего необходимы защитные разряды в регистрах-аккумуляторах и как они используются? С какой целью разрядность операндов в командах, выполняемых операционным блоком, переключается с 40 разрядов на 32 и наоборот?

Вопрос 1.24. При каких операциях над числами и при использовании каких форматов чисел возникает необходимость округления?

Вопрос 1.25. Какие могут возникнуть последствия при отключенном режиме насыщения результатов операций?

Вопрос 1.26. Почему у микропроцессора нет режимов случайного и чередующегося округление чисел?

Вопрос 1.27. По какой причине в устройствах ядра микропроцессора выполнено разделение на блоки для хранения данных и блоки для оперирования данными?

Вопрос 1.28. Почему у операционного устройства имеется множество различных блоков обработки данных: блок манипуляции битами, сдвигатель, двойной умножитель-аккумулятор, арифметико-логический блок?

Вопрос 1.29. Для решения каких задач по обработке сигналов могут потребоваться операции циклического сдвига, извлечения и расширения битовых полей, подсчета числа бит?

Вопрос 1.30. Для чего необходима команда нормализации дробных чисел?

Вопрос 1.31. Почему разрядность операндов команд умножения не может быть равна 32 разрядам?

Вопрос 1.32. Почему перед выполнением умножения множители расширяются с учетом знака до 17 разрядов?

Вопрос 1.33. Требуется ли выполнять операции округления и насыщения при расширении множителей команд умножения до 17 разрядов?

Вопрос 1.34. Почему результат умножения двух дробных чисел сдвигается на один разряд влево?

Вопрос 1.35. Покажите, почему необходим специальный режим коррекции результата умножения чисел 8000h и 8000h? Есть ли другие числа, приводящие к тем же последствиям?

Вопрос 1.36. Возможно ли использование одного и того же регистра-аккумулятора при выполнении двух параллельных команд умножения?

Вопрос 1.37. При решении каких задач может потребоваться фиксация переполнения разрядной сетки в 39 разряде регистра-аккумулятора?

Вопрос 1.38. По какой причине операционное устройство не выполняет генерацию адресов операндов, а устройство адресации – обработку данных?

Вопрос 1.39. Для выполнения каких операций устройство адресации считывает и записывает данные по магистралям данных? Какое устройство генерирует в этом

случае адреса операндов? Возможны ли в этом случае одновременные чтение и запись данных?

Вопрос 1.40. Почему генератор адресов данных не может получать данные с магистралей для чтения данных, но может передавать данные на магистрали для записи?

Вопрос 1.41. Почему разрядность регистра страницы данных периферийных устройств PDP равна 9 разрядам?

Вопрос 1.42. Почему разрядность регистра страницы данных XDP равна 23 разрядам, а не семи? С какой целью старшие девять разрядов регистра DP доступны для чтения-записи через регистр статуса ST0? Почему только девять разрядов, а не другое их число?

Вопрос 1.43. Почему разрядность регистров-указателей равна 23 и не 24 разрядам? Какова причина разделения этих регистров на 16-разрядную младшую часть и 8-разрядную старшую часть?

Вопрос 1.44. Какова причина разделения стека на две части: на системный стек и стек данных? Почему старшая часть регистров-указателей стека одна и та же?

Вопрос 1.45. Почему число регистров длины циклических буферов не равно числу регистров начальных адресов циклических буферов?

Вопрос 1.46. Почему арифметико-логический блок устройства адресации является 16-разрядным при 23-разрядных адресах ячеек памяти?

Вопрос 1.47. Какие операции над адресами выполняет блок генерации адресов данных? Выполняет ли этот блок модификацию содержимого регистров-указателей?

Вопрос 1.48. Укажите, каким образом компилятор языка С передает параметры следующим функциям и как возвращается результат?

```
a: short func_a(long, short, int, short, char, unsigned,
    short *, long long *, int *, unsigned *);
var = func_a(0xD32E0E1D, 0, 1, 2, 3, 4,
    pa, pb, pc, pd);
b: int* func_b(long, long, long long, short, int, unsigned,
    short *, char *, int *, long *);
var = func_b(0x12344321, 0, 1, 2, 3, 4,
    pa, pb, pc, pd);
c: long long func_c(short, char *, int *, unsigned, int, long,
    int *, char *, long, long);
var = func_c(0x2468ABCD, p0, p1, 1, 2, 0x1001,
    p2, p3, 0x98765432, 0x0).
```

Вопрос 1.49. Заданы значения, находящиеся в регистрах XAR0 (010000h), XDP (010000h) и T0 (0004h), а также содержимое области памяти с адресами от 00FFFCh до 010080h:

```
00FFFCh - CCCCh;
00FFFDh - DDDDh;
00FFFEh - EEEEh;
00FFFFh - FFFFh;
010000h - 0000h;
010001h - 1111h;
010002h - 2222h;
010003h - 3333h;
010004h - 4444h;
. . .;
010080h - 8080h.
```

Определите содержимое регистров AC0, AR0 и T0 после выполнения следующих команд:

```
1) MOV *(#x+2), AC0;
2) MOV @(x+1), AC0;
3) MOV @3, AC0;
4) MOV @(x-x+0x80), AC0;
5) MOV *AR0, AC0;
6) MOV *AR0+, AC0;
7) MOV *-AR0, AC0;
8) MOV *(AR0+T0), AC0;
9) MOV *(AR0-T0), AC0;
10) MOV *AR0(T0), AC0;
11) MOV *(AR0+T0B), AC0;
12) MOV *(AR0-T0B), AC0;
13) MOV *AR0(#-1), AC0;
14) MOV *AR0(#2), AC0;
15) MOV *AR0(#0x80), AC0;
16) MOV *+AR0(#3), AC0,
```

где x – метка ячейки памяти с адресом 0010000h.

Вопрос 1.50. Следующая программа написана для умножения комплексных чисел.

```
        .data
x       .word    1, 2     ; Комплексное число x = a+jb = 1+2j
y       .word    7, 8     ; Комплексное число y = c+jd = 7+8j
        .bss     z,2,1,1  ; Выделение памяти для результата
        .global cMul
        .text
_cMul:
        MOV      #x, AR0
        MOV      #Y, CDP
        MOV      #z, AR1
        MOV      #1, T0
        MPY      *AR0, *CDP+, AC0      ; AC0 = a*c
        :: MPY   *AR0(T0), *CDP+, AC1  ; AC1 = b*c
        MAS      *AR0(T0), *CDP+, AC0  ; AC0 = a*c-b*d
        :: MAC   *AR0, *CDP+, AC1      ; AC1 = b*c+a*d
        MOV      pair(LO(AC0)), *AR1+  ; Сохранение результата
        .end
```

Найдите и исправьте ошибки, допущенные в этой программе.

Вопрос 1.51. Для чего и как выполняется разделение программ на модули, запрограммированные на языке С и на языке ассемблера?

Модуль 2
Обработка сигналов и данных

2.1 Обработка данных

Рассмотрим последние два устройства из состава ядра микропроцессора – кэш-команд и буфер команд, или I модуль (рисунок 2.1),[28] а также комплексное функционирование устройств ядра микропроцессора при конвейерном и параллельном выполнении команд.

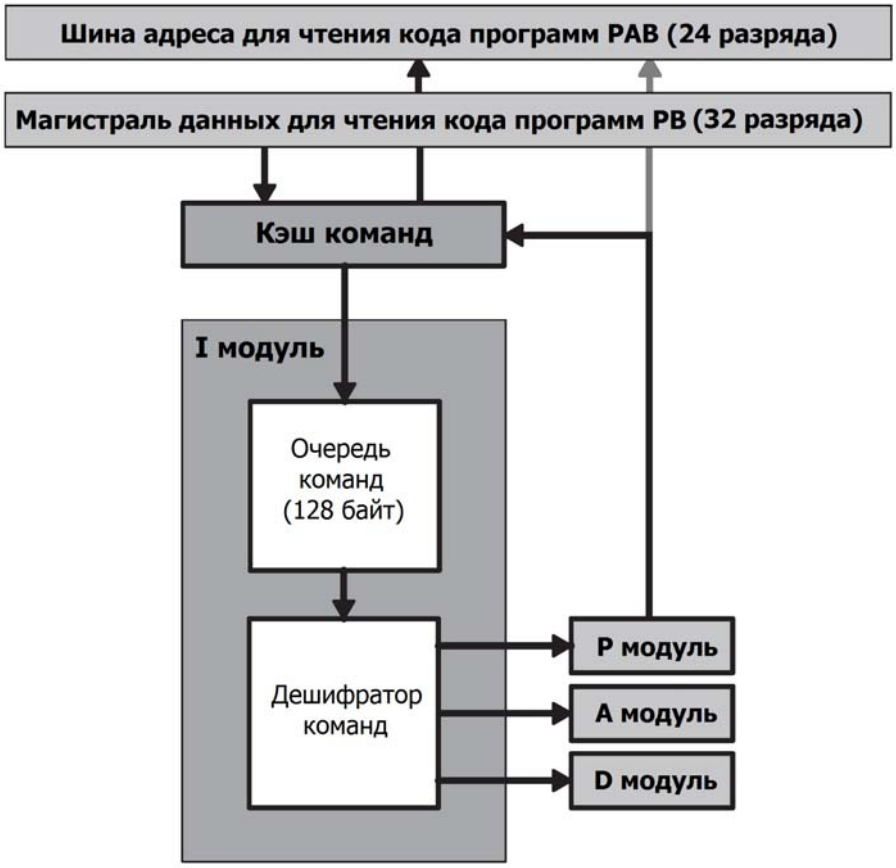


Рисунок 2.1 – Кэш команд и буфер команд

[28] Кэш команд реализован только у микропроцессоров серии C5501, C5502, C5509 и C5510.

2.1.1 Кэш команд

Принцип действия кэша команд заключается в использовании двух свойств потока команд и данных, поступающих в микропроцессор из основной памяти: временной и пространственной локальности.

Временная локальность потока команд и данных характеризуется тем, что если какая-либо команда или порция данных поступила в микропроцессор, то с большой вероятностью эта команда или порция данных потребуется через некоторое время снова.

Пространственная локальность потока команд и данных характеризуется тем, что если какая-либо команда или порция данных поступила в микропроцессор, то с большой вероятностью следующая за ней команда или порция данных также вскоре потребуется.

Появление временной локальности связано с использованием в программах обработки данных различного рода циклов: циклического выполнения блоков команд и циклической обработки массивов данных. В свою очередь появление пространственной локальности связано с последовательным выполнением команд и последовательной обработкой данных.

Кэш команд строится на основе быстродействующей памяти небольшого объёма, которая используется для хранения команд, поступивших в микропроцессор для непосредственного выполнения, а также команд, поступающих в микропроцессор в порядке упреждающего чтения следующих ячеек памяти, размещенных за выбираемыми командами. Использование кэш-памяти позволяет уменьшить время выборки команд, хранящихся в более медленной основной памяти.

Кэш-память хранит копию части данных из основной памяти. Уменьшение времени доступа к памяти программ происходит из-за того, что некоторые команды к моменту их использования микропроцессором оказываются в кэш-памяти. Последнее уменьшает время выборки очередной команды и сокращает общее число обращений к памяти программ.

Кэш-память используется в вычислительных устройствах, у которых имеется существенное различие между временем выборки данных из памяти и временем их обработки. Если данные находятся в основной памяти, то микропроцессор затрачивает некоторое время для их получения. Если требуемые данные уже находятся в кэш-памяти, то они поступают в микропроцессор за меньшее время.

Обычно объем кэш-памяти в сотни раз меньше объема основной памяти. Последнее означает, что возможна ситуация, когда в кэш-памяти не окажется данных, необходимых для обработки. Так как кэш-память в процессе работы всегда заполнена, то новые данные поступают в кэш-память только после замещения (вытеснения) ранее записанных данных. Последнее приводит к тому, что в кэш-памяти накапливаются наиболее часто используемые данные.

Кэш команд состоит из кэш-памяти и кэш-контроллера (рисунок 2.2). Кэш-контроллер управляет кэш-памятью и загружает в неё данные из основной памяти. Для учета локальности потока команд кэш контроллер оперирует не байтами, а блоками данных, соответствующих размеру пакетного цикла чтения-записи основной памяти, равного четырем байтам. По этой причине кэш-память представляет собой совокупность блоков данных фиксированного размера, называемых кэш-строками (англ. Cache Line).

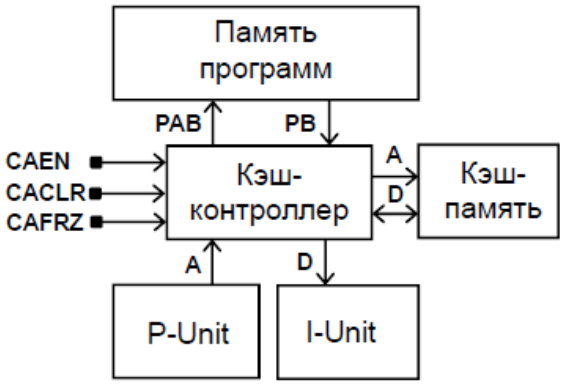


Рисунок 2.2 – Организация кэш-памяти

Когда устройство управления выдает адрес для чтения памяти программ, то этот адрес поступает в кэш-контроллер. Кэш-контроллер определяет, содержатся ли в кэш-памяти данные, соответствующие полученному адресу. Если данные найдены (это событие называется кэш-попаданием, англ. Cache Hit), то кэш-контроллер выдаёт затребованные данные буферу команд. Если же данные не найдены (кэш-промах, Cache Miss), то производится обращение к основной памяти, во время которого микропроцессор простаивает.

Для управления кэш-контролером используются специальные флаги в регистре статуса микропроцессора ST3 (рисунок 1.30):

– CACLR – флаг для очистки кэша команд, который после завершения очистки автоматически сбрасывается в ноль;

– CAEN – флаг разрешения работы кэша команд;

– CAFRZ – флаг блокировки обновления кэша команд, или «замораживание» кэша.

2.1.2 Буфер команд

Буфер команд принимает данные с магистрали данных программы (кэша команд) и выдает команды, которые поступили в эту очередь, устройству управления, устройству адресации и операционному устройству. Устройства ядра микропроцессора получают только те части команд, которые управляют их работой. Буфер команд состоит из очереди команд и дешифратора команд (рисунок 2.1).

Функционирование очереди команд состоит из следующих операций:

– сдвига содержимого очереди команд к началу (в сторону дешифратора команд);

– загрузки восьми байтов (кэш-строки) памяти программ в конец очереди;

– выдача начальных шести байтов очереди команд дешифратору команд;

– очистка очереди команд при ветвлениях в программе (условные и безусловные переходы, вызов и возврат подпрограмм, прерывания).

В свою очередь декодирование команд в предварительном дешифраторе команд происходит следующим образом:

– получение из очереди команд шести байтов кода программы;

– дешифрация начальной (первой) команды и определение ее длины;

– дешифрация следующей команды, определение ее длины и возможности параллельного выполнения с первой командой;

– сдвиг очереди команд на длину одной или двух параллельных команд;

– передача команд устройствам ядра микропроцессора.

На рисунке 2.3 показано упреждающее последовательное заполнение очереди команд для некоторой программы, представленной начальными адресами команд в памяти программ (первая колонка), кодами команд с переменной длиной (вторая колонка), мнемоническими обозначениями команд и их операндов (третья колонка). Операторы на языке С, вызвавшие порождение соответствующего фрагмента кода, показаны в виде заголовков перед каждым таким фрагментом.

Благодаря наличию у очереди команд указателя записи, поступающие из памяти программ очередные восемь байт кода, последовательно записываются в

запоминающее устройство без пропусков. После дешифрации одной или двух параллельно выполняемых команд вся очередь команд сдвигается на длину команд, а указатель записи в очередь команд уменьшается на длину сдвига. Это позволяет организовать выполнение двух независимых процессов, осуществляющих выборку команд и упреждающее заполнение очереди команд.

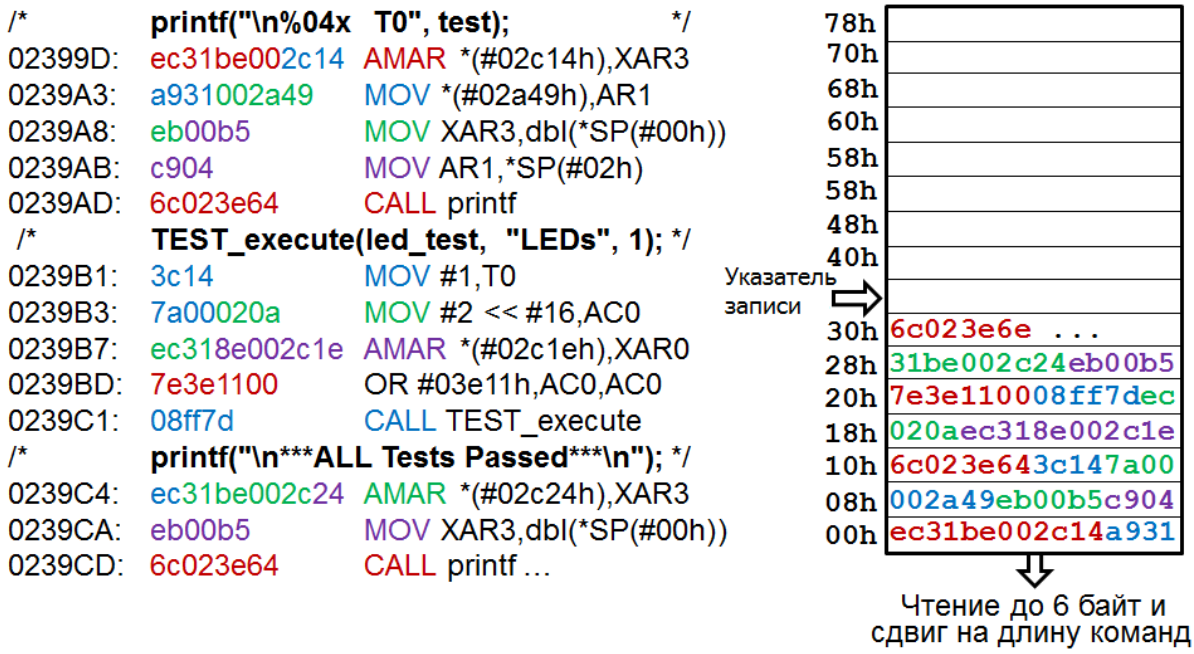


Рисунок 2.3 – Заполнение и сдвиг очереди команд

2.1.3 Конвейеризация команд

Конвейеризация команд увеличивает пропускную способность процессора по выполнению команд, или увеличивает число команд, выполняющихся за единицу времени. При конвейеризации время выполнения каждой отдельной команды не сокращается. Однако благодаря разделению процесса выполнения каждой команды на одинаковые по времени последовательные стадии, появляется возможность совместить во времени выполнение нескольких команд, но на разных их стадиях.

Обработка любой команды микропроцессора можно разделить на этап выборки и этап выполнения команды.

2.1.3.1 Конвейер выборки команд

Выборка команд состоит из следующих стадий (рисунок 2.4):

– формирование и выдача адреса команды на шину адреса PAB (стадия предвыборки 1, англ. Prefetch 1);

– ожидание поступления команды из памяти программ (стадия предвыборки 1, англ. Prefetch 2);

– получение данных с магистрали данных PB и запись их в очередь команд (стадия выборки, англ. Fetch);

– предварительное декодирование команды: определение типа и длины команды, обнаружение параллельных команд (стадия предекодирования, англ. Predecode).

Рисунок 2.4 – Стадии выборки команд

На рисунке 2.5 изображен конвейер выборки команд в момент времени, показанный красной линией, и четыре команды, находящиеся на разных стадиях обработки. Первая команда (самая верхняя) прошла стадии предвыборок PF1 и PF2, а также стадию выборки F, и находится в стадии предекодирования PD. Вторая команда прошла стадии предвыборок PF1 и PF2, и находится в стадии выборки F. Третья команда прошла стадию предвыборки PF1 и находится в стадии предвыборки PF2. В свою очередь последняя четвертая команда находится в стадии предвыборки PF1.

Каждый следующий такт в конвейер поступает новая команда, а команды, находящиеся в конвейере, переходят на новую стадию. В итоге получаем выборку каждой новой команды за один такт при общем 4-тактовом времени выборки одиночной команды.

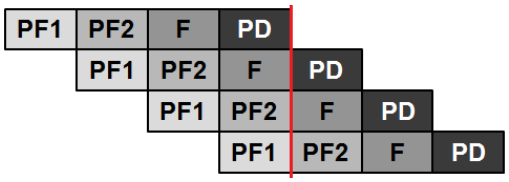


Рисунок 2.5 – Конвейер выборки команд

Таким образом, использование 4-стадийного конвейера выборки команд позволяет ускорить выборку команд до четырех раз.

2.1.3.2 Конвейер выполнения команд

Выполнение команд микропроцессором может быть разделено на следующие стадии (рисунок 2.6):

– получение до шести байт кода одной или двух команды из очереди команд, декодирование команд и передача полей команд устройствам ядра микропроцессора (стадия декодирования, англ. Decode);

– вычисление адресов операндов в устройстве адресации (стадия адресации, англ. Address);

– выдача адресов операндов-источников на шины адреса для чтения данных BAB, CAB и DAB (стадия доступа 1, англ. Access 1);

– ожидание поступления операндов из памяти данных (стадия доступа 2, англ. Access 2);

– получение операндов-источников с магистралей данных BB, CB и DB, а также пересылка содержимого регистров-источников между устройствами (блоками) ядра микропроцессора и вычисление условий ветвления для команд условного выполнения (стадия чтения, англ. Read);

– выполнение команды, изменение содержимого регистров-приемников, установка флагов результата операций (стадия выполнения, англ. Execute);

– запись данных в регистры, отображаемые в память, или выдача адреса операндов-приемников на шины адреса для записи данных EAB и FAB (стадия записи, англ. Write).

– если необходимо, запись данных операндов-приемников на магистрали для записи данных EB и FB (стадия записи+, англ. Write+).

Рисунок 2.6 – Стадии выполнения команд

На рисунке 2.7 изображен конвейер выполнения команд в момент времени, показанный красной линией. В конвейер загружено восемь команд, находящихся на разных стадиях выполнения. Первая команда AAD прошла все стадии и завершила свое исполнение, так как стадия W+ для выполнения этой команды не требуется. Вторая команда MOV прошла стадии D, AD, AC1, AC2, R, X и W и находится на стадии записи W+, т.е. выполнена на 87 %. Последняя команда MOV прошла стадию декодирования и выполнена на 12,5 %. Следующая команда MOV еще не поступила в конвейере и выполнена на 0 %.

Как и у конвейера выборки команд, каждый такт в конвейер выполнения команд поступает новая команда, а команды, находящиеся в конвейере, переходят на следующую стадию. В итоге получаем, что при общем времени выполнения каждой отдельной команды, равном восьми тактам, команды в конвейере выполняются за один такт.

Таким образом, использование 8-стадийного конвейера выполнения команд позволяет ускорить обработку команд в восемь раз. Если учесть, что конвейер выборки команд последовательно соединен с конвейером выполнения команд, то суммарное ускорение обработки потока команд равно 12.

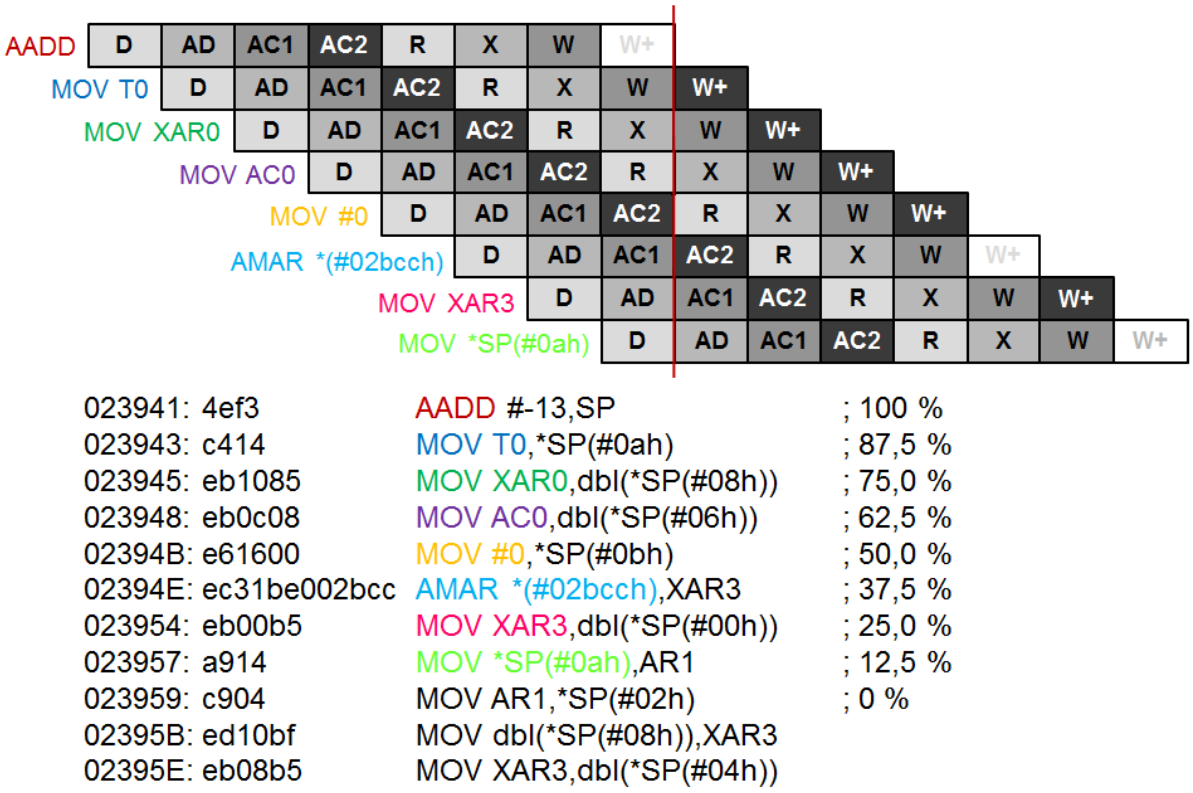


Рисунок 2.7 – Конвейер выборки команд

В итоге получается, что выполнение одной команды микропроцессора происходит в течение 12 тактов. Однако 12-стадийный конвейер команд позволяет выполнять одну команду за один такт при 11-тактовой начальной задержке, связанной с начальным заполнением объединенного конвейера команд.

2.1.3.3 Перезагрузка конвейера

Обработка данных на каждой стадии конвейера занимает один такт и производится устройствами и блоками ядра микропроцессора. При полной загрузке конвейера в каждом такте его работы будет выполняться очередная команда.

Однако при выполнении программ возникают события, при которых конвейер должен быть очищен и загружен новой последовательностью команд. К событиям, которые вызывают перезагрузку конвейера, относятся:

– выполнение команд безусловных и условных переходов B и BCC;

– изменение последовательности выполнения команд, вызванное командами безусловного и условного вызова подпрограмм CALL, CALLCC, а также командой программного прерывания INTR;

– изменение последовательности выполнения команд, вызванное командами безусловного и условного возврата из подпрограмм RET, RETCC, а также командой возврата из прерывания RETI;

– внешние прерывания по сигналам с выводов микропроцессора INT0, INT1;

– внутренние немаскируемые прерывания по сигналу NMI от схем контроля микропроцессора;

– сброс микропроцессора по внешнему сигналу RESET или командой RESET.

Перезагрузка конвейера приводит к потере производительности микропроцессора. Это связано с тем, что для выполнения первой команды, поступившей в пустой конвейер, требуется прохождение этой команды по всем его стадиям. Например, перезагрузка 12-стадийного конвейера рассматриваемого микропроцессора вносит задержку, равную 11 тактам.

2.1.3.4 Конфликты в конвейере

Помимо событий, вызывающих перезагрузку конвейера, возможны также ситуации, при которых в конвейере возникают конфликты, связанные с зависимостью команд по данным. Зависимость команд по данным возникает в том случае, когда одна команда, загруженная в конвейер, использует результат выполнения другой команды, также находящейся в конвейере.

Рассмотрим пример конфликтной ситуации типа «запись до чтения», возникающей в конвейере при зависимости двух смежных команд по содержимому регистров (рисунок 2.8):

MOV AR0, *AR1 – сохранение содержимого регистра AR0 в ячейке памяти с адресом, находящимся в регистре XAR1;

MOV *+AR0, AR2 – запись содержимого ячейки памяти с адресом, равным XAR0+1 в регистр AR2, и сохранение этого адреса в качестве нового содержимого регистра XAR0.

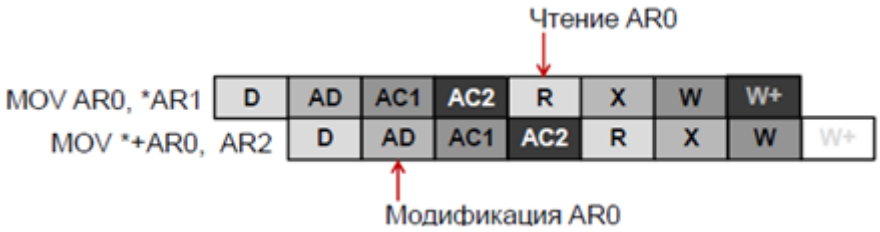


Рисунок 2.8 – Конфликт типа «запись до чтения»

Из рисунка видно, что вторая команда модифицирует содержимое регистра AR0 до того, как его содержимое будет записано первой командой в адресуемую ячейку памяти.

Другой тип конфликтной ситуации называется «одновременный доступ к памяти» и возникает в том случае, когда две смежные команды выполняют запись операндов в память (рисунок 2.9а):

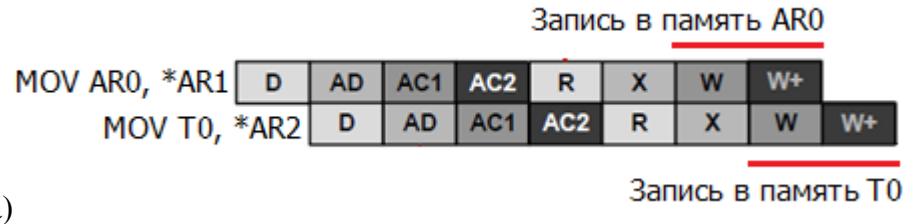
– MOV AR0, *AR1 – сохранение содержимого дополнительного регистра AR0 в ячейке памяти с адресом, находящимся в регистре XAR1;

– MOV T0, *AR2 – запись содержимого временного регистра T0 в ячейку памяти с адресом, равным XAR2.

Эта конфликтная ситуация также возникает, когда две смежные команды выполняют чтение операндов из памяти (рисунок 2.9б):

– MOV *AR1, AR0, – загрузка дополнительного регистра AR0 содержимым ячейки памяти с адресом, находящимся в регистре XAR1;

– MOV *AR2, T0 – загрузка временного регистра T0 содержимым ячейки памяти с адресом, находящимся в регистре XAR2.



а)

б)

Рисунок 2.9 – Конфликты типа «одновременный доступ к памяти»

Как видно из рисунка, стадии чтения и записи операндов команд перекрываются во времени, что вызывает конфликт одновременного доступа к памяти.

В ядро микропроцессора включен блок защиты конвейера, который обнаруживает конфликты и добавляет холостые циклы в работу конвейера, исключающие зависимость команд по данным (рисунок 2.10). После загрузки в конвейер между зависимыми командами трех холостых команд NOP (англ. No Operation), зависимость команд по данным устраняется и конфликт исчезает.

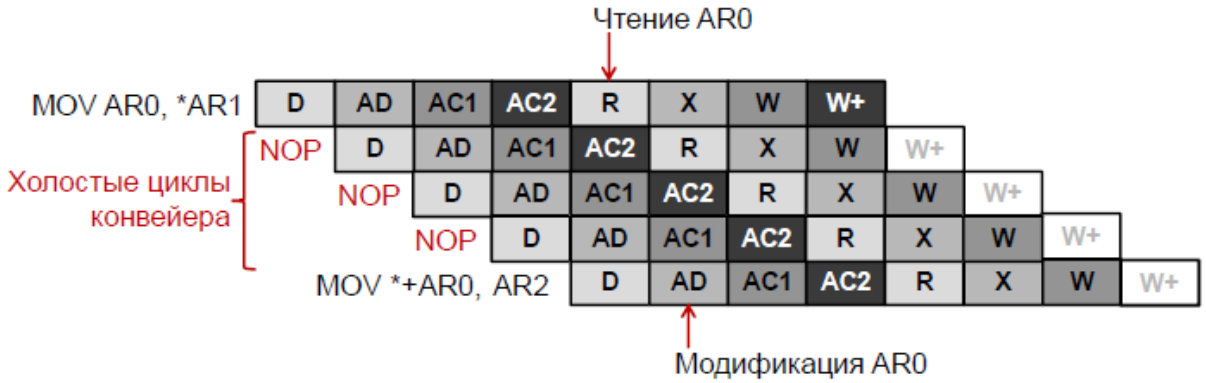


Рисунок 2.10 – Холостые циклы конвейера

Блок защиты конвейера обнаруживает и устраняет возникающие конфликты в конвейере. Однако это приводит к замедлению работы конвейера и уменьшению производительности микропроцессора. Поэтому при программировании задач по цифровой обработке сигналов рекомендуется запускать программу или ее критическую часть в среде симулятора и контролировать число тактов, потребовавшихся для выполнения команд.[29] Если число тактов окажется больше одного, то это сигнализирует о конфликтах в конвейере.

Для минимизации числа холостых циклов в конвейере используется несколько приемов:

– перестановка команд, которая не изменяет алгоритм обработки данных, однако разносит взаимосвязанные команды на большее расстояние в памяти программ;

– преобразование команд обращения к операндам в памяти программ в команды пересылки операндов в регистры микропроцессора и использование этих регистров для доступа к операндам;

– перемещение данных, вызывающих конфликт одновременного доступа к памяти, в двухвходовую память;

[29] При использовании эмулятора возможность контролировать число тактов, потребовавшихся для выполнения команд, отсутствует.

– использование повторения одиночных команд и повторения блоков команд, размещаемых в буфере команд и не загружающих интерфейс доступа к памяти циклами выборки команд.

2.1.4 Распараллеливание команд

Микропроцессор поддерживает четыре типа параллельного выполнения команд:

– параллелизм, встроенный в команду;

– параллелизм команд, задаваемый пользователем;

– комбинированный параллелизм.

2.1.4.1 Встроенный параллелизм

При встроенном параллелизме две части одной команды, имеющие аналоги в наборе команд микропроцессора, соединяются двойным двоеточием и образуют единую команду, например:

MPY *AR0, *CDP, AC0;

:: MPY *AR1, *CDP, AC1,

где первая команда для умножения содержимого ячейки памяти, адресуемой через регистр-указатель XAR0, на содержимое ячейки памяти, адресуемой через регистр-указатель XCDP, использует первый умножитель операционного устройства, а вторая команда использует второй умножитель для параллельного выполнения второй операции умножения содержимого ячейки памяти, адресуемой через регистр-указатель XAR1, на содержимое ячейки памяти, адресуемой через тот же регистр-указатель XCDP.

2.1.4.2 Пользовательский параллелизм

Для параллельного выполнения двух команд в форматах многих команд предусмотрено специальное одноразрядное поле, где задается флаг параллельности E. На языке ассемблера установка этого флага выражается двумя вертикальными чертами, записываемыми перед мнемоникой команды, например:

ADD AC0, AC1;

|| XOR AR2, CDP.

В этом примере первая команда складывает содержимое регистров-аккумуляторов AC0 и AC1, помещает результат в регистр-аккумулятор AC1, для чего задействуется арифметико-логический блок операционного устройства. Вторая команда, выполняемая параллельно с первой, реализует операцию поразрядного логического сложения по модулю два содержимого дополнительного регистра AR2 и

регистра коэффициентов CDP, для чего используется арифметико-логический блок устройства адресации.

2.1.4.3 Комбинированный параллелизм

Комбинированный параллелизм возникает тогда, когда параллельно выполняются две команды: первая команда со встроенным параллелизмом, а вторая команда с параллелизмом, задаваемым установкой флага параллельности E:

```
MPY *AR0, *CDP, AC0;
:: MPY *AR1, *CDP, AC1;
|| MOV #5, AR2.
```

Все три команды могут выполняться параллельно, что ускоряет работу программы. В первой команде используется первый умножитель операционного устройства, во второй команде – второй умножитель, а третья команда – команда загрузки в дополнительный регистр AR2 константы 5, выполняется устройством адресации.

2.1.4.4 Условия параллелизма

Для пары параллельно выполняемых команд должны выполняться следующие условия:

– суммарная длина команд должна быть не более шести байт (команды должны декодироваться совместно);

– не должно возникать конфликтов по использованию ресурсов (см. пп. 2.1.4.5);

– одна из команд должна иметь поле для задания флага параллельности E;

– не должны использоваться режимы адресации с 16-разрядными непосредственными данными: *abs16(#k16), *(#k23), port(#k16), *ARx(K16), *+ARx(K16), *CDP(K16) и *+CDP(K16);

– одной из команд не могут быть команды условного перехода BCC и вызова подпрограмм CALLCC с абсолютным 24-разрядным адресом, команды прерывания INTR и отладочного прерывания TRAP, команда сброса RESET;

– одной из команд не могут быть команды, содержащие префиксы обращения в адресное пространство ввода-вывода port(), изменения режима циклической адресации .CR и .LR, обращения к регистрам, отображаемым в память mmap().

– в одной стадии конвейера не может быть выполнено более одной записи в регистр или ячейку памяти.

2.1.4.5 Параллельные операции

На рисунке 2.11 изображена таблица операций, совместимых для параллельного выполнения, где знаком x показаны пары операций, приводящие к конфликтам по используемым ресурсам.

			A модуль				D модуль							P модуль		
			АЛУ	Перестановка	Загрузка	Сохранение	АЛУ	Сдвиг	Умножение	Загрузка	Сохранение	Сдвиг и сохр.	Перестановка	Управление	Загрузка	Сохранение
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
A модуль	АЛУ	1	X													
	Перестановка	2		X												
	Загрузка	3														
	Сохранение	4														
D модуль	АЛУ	5					X	X	X							
	Сдвиг	6					X	X	X			X				
	Умножение	7					X	X	X							
	Загрузка	8														
	Сохранение	9														
	Сдвиг и сохр.	10						X				X				
	Перестановка	11											X			
P модуль	Управление	12												X		
	Загрузка	13														
	Сохранение	14														

Рисунок 2.11 – Параллельные операции

2.2 Обработка сигналов

Общая схема обработки сигналов приведена на рисунке 2.12. Из рисунка видно, что входной сигнал непрерывно поступает во входной канал, где подвергается предварительной аналоговой обработке: усилению и фильтрации. Аналогово-цифровое преобразование выполняется с некоторой частотой, называемой частотой дискретизации, в результате чего получается дискретный входной сигнал, изменяющийся в дискретные моменты времени и принимающий дискретные значения.

Устройство цифровой обработки сигналов объединяет смежные во времени отсчеты дискретного сигнала $x(\tau)$ в массив конечной длины N, и выполняет его преобразование в массив отсчетов выходного дискретного сигнала $y(\tau)$.

Далее элементы массива выходного сигнала $y(\tau)$ с задержкой, равной N интервалам дискретизации, последовательно передаются на вход цифро-аналогового преобразователя, и полученный с выхода сигнал подвергается заключительной аналоговой фильтрации и усилению.

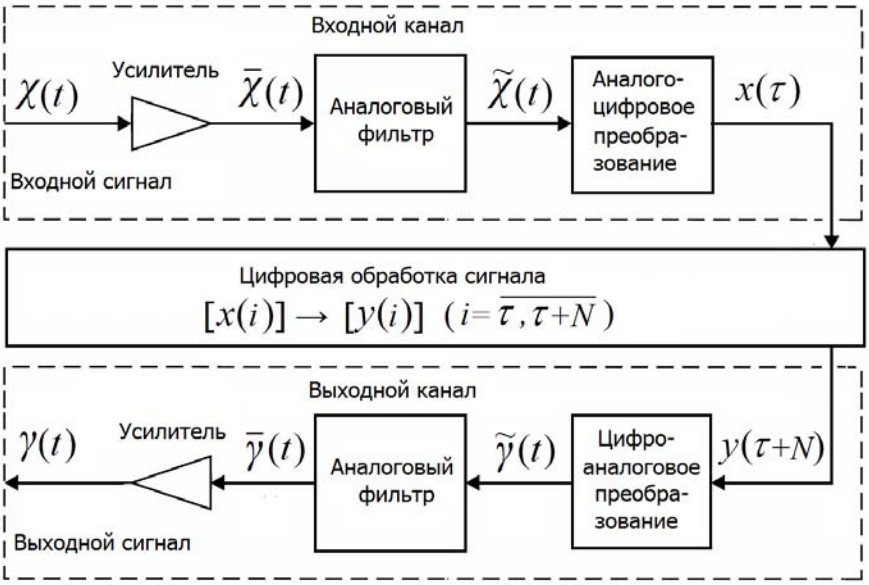


Рисунок 2.12 – Аналоговая и цифровая обработка сигналов

2.2.1 Задачи цифровой обработки сигналов

Под сигналом $\mathbf{X}$ понимают физический процесс, протекающий во времени и отражающий состояние некоторого объекта наблюдения.[30]

Аналоговый сигнал рассматривается как непрерывная функция χ времени t:

$$\chi(t) \in [\chi_{\min}, \chi_{\max}]; \quad t \in [t_{\min}, t_{\max}],$$

где $[\chi_{\min}, \chi_{\max}]$ – конечный интервал изменения значения некоторого параметра сигнала $\mathbf{X}$, $[t_{\min}, t_{\max}]$ – конечный интервал времени t, в течение которого наблюдается изменение параметра χ сигнала $\mathbf{X}$.[31]

[30] Физический сигнал характеризуется множеством параметров, поэтому он обозначен как вектор $\mathbf{X}$.
[31] В связи с невозможностью практической реализации сигналов, имеющих разрывы первого и второго рода, и невозможностью наблюдения сигналов бесконечное время, аналоговые сигналы, заданные кусочно-непрерывными функциями и на бесконечных интервалах далее не рассматриваются.

Дискретный сигнал x (рисунок 2.13) рассматривается как последовательность отсчетов (выборок) аналогового сигнала χ, взятых в дискретные моменты времени τ и принимающих дискретные значения x:

$$x(\tau) \in \overline{0, L-1} \quad (\tau = \overline{0, N-1}),$$

где физическое χ и дискретное x значение параметра сигнала $\mathbf{X}$, физическое t и дискретное время τ связаны соотношениями:

$$t = t_{\min} + \tau\, T; \quad T = \frac{t_{\max} - t_{\min}}{N},$$

$$\chi = \chi_{\min} + x\, D \; ; \quad D = \frac{\chi_{\max} - \chi_{\min}}{L},$$

где T – шаг квантования во времени, N – число отсчетов сигнала, D – шаг квантования по уровню, L – число уровней квантования.

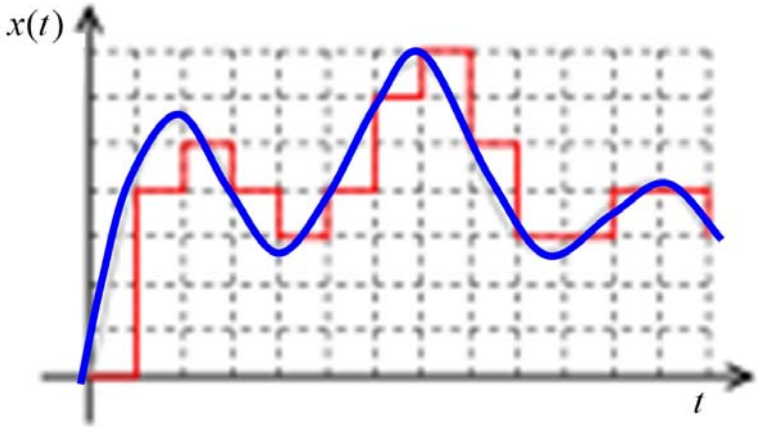


Рисунок 2.13 – Аналоговый и дискретный сигнал

2.2.1.1 Дискретизация аналоговых сигналов

Для получения из аналогового сигнала $\chi(t)$ его дискретного представления $x(\tau)$ выполняется квантование сигнала $\chi(t)$ во времени (рисунок 2.14):

$$\chi(t) \in [\chi_{\min}, \chi_{\max}],\ t \in [t_{\min}, t_{\max}] \Rightarrow \chi(\tau) \in [\chi_{\min}, \chi_{\max}]\ (\tau = \overline{0, N-1}),$$

и квантование сигнала $\chi(\tau)$ по уровню (рисунок 2.15):

$$\chi(\tau) \in [\chi_{\min}, \chi_{\max}] \Rightarrow x(\tau) \in \overline{0, L-1} \quad (\tau = \overline{0, N-1}).$$

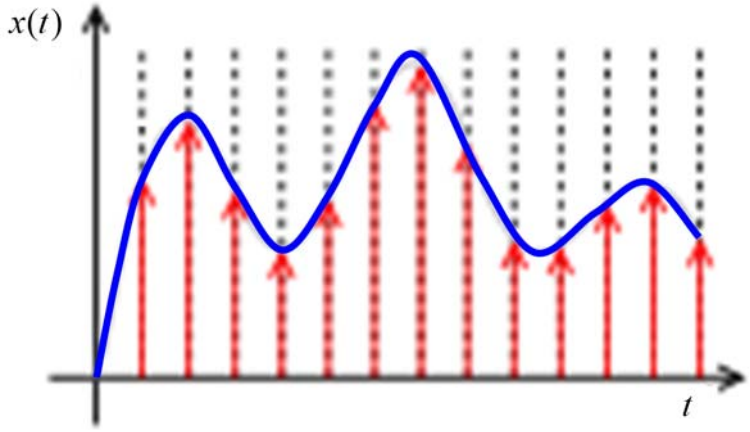


Рисунок 2.14 – Квантование во времени

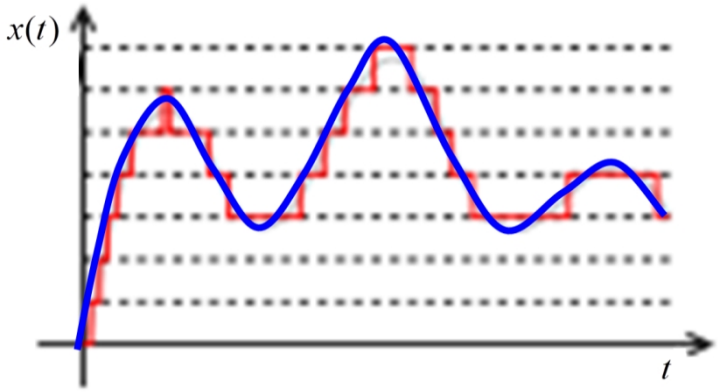


Рисунок 2.15 – Квантование по уровню

2.2.1.2 Цифровая обработка сигнала

Цифровая обработка сигнала заключается в преобразовании последовательности дискретных отсчетов входного сигнала $x(\tau)$:

$$x(0),\ x(1),\ \ldots,\ x(N-1),$$

в последовательность дискретных отсчетов выходного сигнала $y(\tau)$:

$$y(0),\ y(1),\ \ldots,\ y(N-1),$$

где N – число обрабатываемых отсчетов.[32]

2.2.1.3 Передискретизация

Передискретизация (англ. resampling) заключается в изменении частоты дискретизации сигнала. Передискретизации широко применяются при обработке звуковых сигналов, радиосигналов и изображений.

[32] В общем случае число отсчетов входного сигнала и число отсчетов выходного сигнала может быть различным.

Отсчёты сигнала, соответствующие новой частоте дискретизации, вычисляются по уже имеющимся отсчётам:

$$x(0),\ x(1),\ \ldots,\ x(N-1) \Rightarrow y(0),\ y(1),\ \ldots,\ y(M-1),$$

где N – число отсчетов входного сигнала, M – число отсчетов выходного сигнала.

Повышение частоты дискретизации называется интерполяцией ($N < M$), а понижение – децимацией ($N > M$).

2.2.1.4 Функциональные преобразования

Наиболее простой обработкой сигнала является его функциональное преобразование:

$$y(\tau) = f(x(\tau)) \quad (\tau = \overline{0, N-1}),$$

где f – некоторая дискретная функция преобразования отсчетов, например: ограничение сигнала по уровню, обнуление малых значений, нелинейное масштабирование, амплитудная модуляция, и т.п.

Другой важный случай – модуляция сигнала, или изменение параметров входного сигнала $x(\tau)$ модулирующим сигналом $m(\tau)$:

$$y(\tau) = f(x(\tau), m(\tau)) \quad (\tau = \overline{0, N-1}).$$

Вычислительная сложность функционального преобразования сигналов обычно равна nN операций, где n – некоторая константа, равная среднему числу операций, необходимых для вычисления функции f.

2.2.1.5 Корреляция

Корреляция $r(\tau)$ двух сигналов $x(i)$ $(i = \overline{0, N-1})$ и $y(j)$ $(j = \overline{0, M-1})$ применяется для определения степени независимости одного сигнала от другого:

$$r(\tau) = \frac{1}{N} \sum_{k=0}^{K-\tau-1} x(\tau + k) \times y(k) \quad (\tau = \overline{0, K-1}),$$

где K – число отсчетов корреляции сигналов, $K = N + M - 1$.

При вычислении корреляции входные сигналы рассматривается как периодические:[33]

$$x(i \pm N) = x(i) \quad (i = \overline{0, N-1});$$

$$y(j \pm M) = y(j) \quad (j = \overline{0, M-1}).$$

Вычислительная сложность нахождения корреляции – $(N + M)/2$ операции умножения со сложением и $N + M - 1$ операция деления.

[33] Для реализации периодического входного сигнала его отсчеты хранятся в циклическом буфере.

Корреляция применяется для расчета спектральной плотности энергии сигнала, для детектирования и оценки периодических сигналов в шуме, для определения отношения сигнал-шум у периодического зашумленного сигнала, для корреляционного детектирования сигналов, для определения импульсной характеристики устройств цифровой обработки сигналов.

2.2.1.6 Свертка

Свертка сигнала $x(\tau)$ с импульсной характеристикой $h(i)$ $(i = \overline{0, H-1})$ некоторого линейного устройства цифровой обработки сигналов позволяет получить выходной сигнал устройства $y(\tau)$ по его входному сигналу $x(\tau)$:

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times x(\tau - i) \quad (\tau = \overline{0, N-1}).$$

Вычислительная сложность свертки – NH операций умножения со сложением. При вычислении свертки входной сигнал рассматривается как периодический,

$$x(-\tau) = x(N - \tau) \quad (\tau = \overline{0, N-1}).$$

Свертка используется при идентификации устройств цифровой обработки сигналов, для вычисления входного сигнала устройства при известном его выходном сигнале, при блочной нерекурсивной цифровой фильтрации и свёрточном кодировании сигналов.

2.2.1.7 Фильтрация

При цифровой фильтрации выполняется более сложная обработка сигналов:

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times x(\tau - i) - \sum_{j=1}^{G} g(j) \times y(\tau - j) \quad (\tau = \overline{0, N-1}), \qquad (2.1)$$

где $x(\tau)$ – входной сигнал, $y(\tau)$ – выходной сигнал, $h(i)$ и $g(j)$ – импульсная характеристика линейной и нелинейной части устройства цифровой обработки сигналов.

Если выполняется условие $G > 0$, то цифровой фильтр является рекурсивным, или фильтром с бесконечной импульсной характеристикой (БИХ-фильтром), а при выполнении условия $G = 0$ – нерекурсивным, или фильтром с конечной импульсной характеристикой (КИХ-фильтром):

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times x(\tau - i) \quad (\tau = \overline{0, N-1}). \qquad (2.2)$$

Вычислительная сложность фильтрации – $N(H + G)$ операций умножения со сложением. Необходима также дополнительная память в объеме $H + G - 1$ ячейка:

– $H-1$ ячейка – для хранения предыдущих отсчетов входного сигнала $x(\tau)$: $x(-1)$, $x(-2)$, …, $x(-H+1)$;

– G ячеек – для хранения предыдущих отсчетов выходного сигнала $y(\tau)$: $y(-1)$, $y(-2)$, …, $y(-G)$.

2.2.1.8 Спектральная обработка

При спектральной обработке входной сигнал $x(\tau)$ представляется в виде линейной комбинации ортогональных сигналов $s_i(\tau)$,

$$x(\tau) = \sum_{i=0}^{N-1} c(i) \times s_i(\tau) \quad (\tau = \overline{0, N-1}),$$

таких, что

$$\sum_{\tau=0}^{N-1} s_i(\tau) \times s_j(\tau) = \delta_{ij} \quad (i, j = \overline{0, N-1}),$$

где δ_{ij} – функция Кронекера (равна единице при равенстве i и j, и нулю – в противном случае), $c(i)$ – коэффициенты спектрального разложения, или спектр сигнала,

$$c(i) = \sum_{\tau=0}^{N-1} x(\tau) \times s_i(\tau) \quad (i = \overline{0, N-1}). \qquad (2.3)$$

После преобразований спектральных коэффициентов $c(i)$ функцией f,

$$c'(i) = f(c(i)) \quad (i = \overline{0, N-1}), \qquad (2.4)$$

выполняется обратный переход из спектральной области во временную,

$$y(\tau) = \sum_{i=0}^{N-1} c'(i) \times s_i(\tau) \quad (\tau = \overline{0, N-1}), \qquad (2.5)$$

в результате чего получается выходной сигнал $y(\tau)$.

В качестве спектральных функций используются гармонические функции, функции Уолша, Адамара, Хаара, Радемахера, Хартли и Бесселя, вейвлет-функции, ортогональные многочлены Чебышева, Якоби, Лежандра, Эрмита, Легерра, Гегенбауэра и др.

Вычислительная сложность спектральной обработки сигналов самая высокая и равна $2N^2$ операций умножения со сложением – для вычисления прямого (2.3) и обратного (2.5) преобразования, и nN операций – для преобразования спектра сигнала (2.4), где n – некоторая константа, равная среднему числу операций, необходимых для

вычисления функции f.[34] Требуется также дополнительная память для хранения исходного $c(i)$ и результирующего спектра $c'(i)$ объемом $2N$ ячеек.[35]

Спектральная обработка сигналов применяется для аппроксимации сигналов, при частотном и фазовом их анализе, при параметрической идентификации и диагностике динамических объектов, в задачах анализа и распознавания образов и т.п.

2.2.1.9 Восстановление сигнала по отсчетам

После получения выходного дискретного сигнала $y(\tau)$ выполняется восстановление его аналогового представления $\gamma(t)$:

$$y(\tau) \in \overline{0, K-1} \ (\tau = \overline{0, N-1}) \ \Rightarrow \gamma(t) \in [\gamma_{\min}, \gamma_{\max}], \ t \in [t_{\min}, t_{\max}],$$

которое сводится к использованию различных интерполяционных процедур.

Для надежного восстановления аналогового сигнала по его дискретному представлению необходимо соблюдать следующие условия:

$$T < \frac{\chi_{\max} - \chi_{\min}}{|\chi'(t)|_{\max}},$$

где $|\chi'(t)|_{\max}$ – максимальное значение модуля первой производной аналогового сигнала $\chi(t)$.[36]

2.2.2 Нерекурсивный фильтр

Нерекурсивные фильтры (2.2) наиболее часто используются в телекоммуникационных приложениях, где требуется устойчивая фильтрация сигналов с заранее известными параметрами. На рисунке 2.16 показана структура нерекурсивного фильтра, где блоки z^{-1} реализуют задержку сигнала на один шаг квантования по времени T.

[34] Для прямого и обратного преобразования Фурье существуют быстрые алгоритмы вычисления спектральных коэффициентов со сложностью, равной $N \log N$, где N – степень двойки.

[35] Объем дополнительной памяти может быть сокращен вдвое, до N ячеек, если хранение преобразованных спектральных коэффициентов осуществляется в том же массиве, что и исходных.

[36] Максимальное значение модуля первой производной может быть оценено по амплитуде A сигнала и его максимальной частоте F, $|\chi'(t)|_{\max} = AF$.

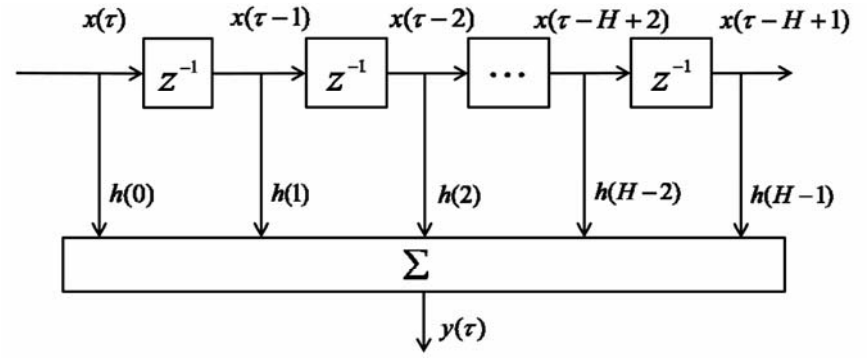


Рисунок 2.16 – Нерекурсивный фильтр

Частным случаем нерекурсивных фильтров является фильтр с линейной фазовой характеристикой, или симметричный КИХ-фильтр, реализующий изменение фазы сигнала пропорционально его частоте. Такое преобразование сигнала обеспечивается симметричной импульсной характеристикой фильтра $h(i)$,

$$h(i) = h(H-1-i) \quad (i = \overline{0, H-1}),$$

где H – общее число коэффициентов фильтра.

Другой частный случай – нерекурсивный фильтр с антисимметричной импульсной характеристикой,

$$h(i) = -h(H-1-i) \quad (i = \overline{0, H-1}).$$

Применение антисимметричных КИХ-фильтров связано с реализацией преобразования Гильберта, которое сдвигает положительные частоты сигнала на плюс 90 градусов, а отрицательные – на минус 90 градусов.

Поскольку симметричный и антисимметричный фильтры имеют только $H/2$ различных коэффициентов, вычисление одного отсчета выходного сигнала требует всего $H/2$ операций умножениями с двойным сложением:

$$y(\tau) = \sum_{i=0}^{H/2-1} h(i) \times \big(x(\tau - i) + x(\tau - H + i + 1)\big) \quad (\tau = \overline{0, N-1}), \qquad (2.6)$$

где N – число отсчетов входного $x(\tau)$ и выходного сигналов $y(\tau)$.[37]

Для реализации симметричной и антисимметричной нерекурсивной фильтрации сигналов используется специальные команды микропроцессора FIRSADD и FIRSSUB. Команда симметричной (кососимметричной) фильтрации FIRSADD (FIRSSUB) имеет 5 операндов:

[37] Если число коэффициентов фильтра будет нечетным, то у формулы (2.6) появится одно дополнительное слагаемое вида $h(k) \times x(\tau - k)$, где $k = (H+1)/2$.

– Xmem – текущее значение входного отсчета, адресуемое через дополнительные регистры-указатели;

– Ymem – симметричное текущему значение входного отсчета, адресуемое через дополнительные регистры-указатели;

– Cmem – текущий коэффициент фильтра, адресуемый через регистр-указатель коэффициентов CDP;

– ACx – регистр-аккумулятор AC0-AC3, содержащий сумму (разность) текущего и симметричного текущему отсчетов входного сигнала;

– ACy – регистр-аккумулятор AC0-AC3, накапливающий значение выходного отсчета.

Команда выполняет две параллельные операции умножение с накоплением и сложение (вычитание):

ACy = ACy + (ACx * Cmem);

ACx = (Xmem << #16) + (Ymem << #16) (ACx = (Xmem << #16) – (Ymem << #16)).

Пример программы, реализующей симметричный блочный нерекурсивный цифровой фильтр, приведен на листинге 2.1.[38] Предполагается, что входные и выходные отсчеты сигнала, а также коэффициенты фильтра хранятся в памяти в формате Q1.15.

Листинг 2.1 – Симметричная нерекурсивная фильтрация

```
; Исходные данные
.ASG     #010100h, X       ; Адрес входного массива
.ASG     #020200h, Y       ; Адрес выходного массива
.ASG     #030300h, C       ; Адрес массива коэффициентов
.ASG     32, N             ; Число отсчетов сигнала
.ASG     8, H              ; Число коэффициентов фильтра
; Задание режима работы микпропроцессора
BSET     FRCT, ST1         ; Дробный формат чисел
BSET     SXMD, ST1         ; Знаковый формат операндов
BCLR     M40, ST1          ; 32-разрядные операции
BCLR     SATD, ST1         ; Отключение насыщения при вычислениях
BCLR     RDM, ST2          ; Отключение банковского округления
BCLR     SMUL, ST3         ; Отключение насыщения при умножении
BCLR     SATA, ST3         ; Отключение насыщения при адресации
; Загрузка регистров-указателей
AMOV     #X, XAR1          ; Адрес входного буфера
AMOV     #Y, XAR3          ; Адрес выходного буфера
AMOV     #C, XCDP          ; Адрес буфера коэффициентов
```

[38] С целью упрощения в листинге приведена реализация фильтра, не имеющего дополнительного буфера для запоминания и хранения предыдущих отсчетов входного сигнала $x(\tau)$, т.е. входной сигнал предполагается периодическим, $x(\tau) = x(\tau - N)$, с периодом, равным NT, где N – длина входного буфера, T – на шаг квантования во времени.

Листинг 2.1 – Симметричная нерекурсивная фильтрация

```
; Настройка циклического буфера коэффициентов
MOV      #H, BKC           ; Размер буфера коэффициентов
BSET     CDPLC, ST2        ; Флаг циклической адресации CDP
MOV      mmap(CDP), BSAC   ; Базовый адрес буфера коэффициентов
MOV      #0, CDP           ; Начальное смещение коэффициентов
; Настройка циклического буфера входных отсчетов
MOV      #N, BK03          ; Размер входного буфера
BSET     AR0CL, ST2        ; Флаг циклической адресации AR0
BSET     AR1CL, ST2        ; Флаг циклической адресации AR1
MOV      mmap(AR0), BSA01  ; Базовый адрес входного буфера
MOV      #0, AR0           ; Смещение первого отсчета
MOV      #N-1, AR1         ; Смещение последнего отсчета
; Настройка внешнего и внутреннего циклов
MOV      #(N-1), BRC0      ; Счетчик повторения внешнего цикла
MOV      #(H-1), CSR       ; Счетчик повторения внутреннего цикла
MOV      XAR1, XAR2        ; Адрес первого входного отсчета
ADD      #(N-1), AR2       ; Адрес последнего входного отсчета
MOV      #(H-1), T0        ; Смещение следующей итерации
; Цикл вычисления выходных отсчетов сигнала
REPLOCAL {
     ; Начальные установки внешнего цикла
     MOV      #0, AC0
     ; Начало первой итерации внутреннего цикла
     ADD      *AR0+, *AR1-, AC1
     ; Внутренний цикл
     REP      CSR
          FIRSADD  *AR0+, *AR1-, *CDP+, AC1, AC0
     ; Окончание последней итерации внутреннего цикла
     MACMR    *CDP+, AC1, AC0
     ; Запись выходного отсчета
     MOV      HI(AC0), *AR3+
     ; Смещение указателей для следующей итерации
     AMAR     *(AR0-T0)
     AMAR     *(AR1+T0)
}
```

2.2.3 Рекурсивный фильтр

Рекурсивные фильтры (2.1) при прочих равных условиях являются более эффективными, чем нерекурсивные, так как имеют меньшую вычислительную сложность и требуют меньший объем дополнительной памяти. На рисунке 2.17 показана структура рекурсивного фильтра, где блоки z^{-1}, как и ранее, реализуют задержку сигнала на один шаг квантования по времени T.

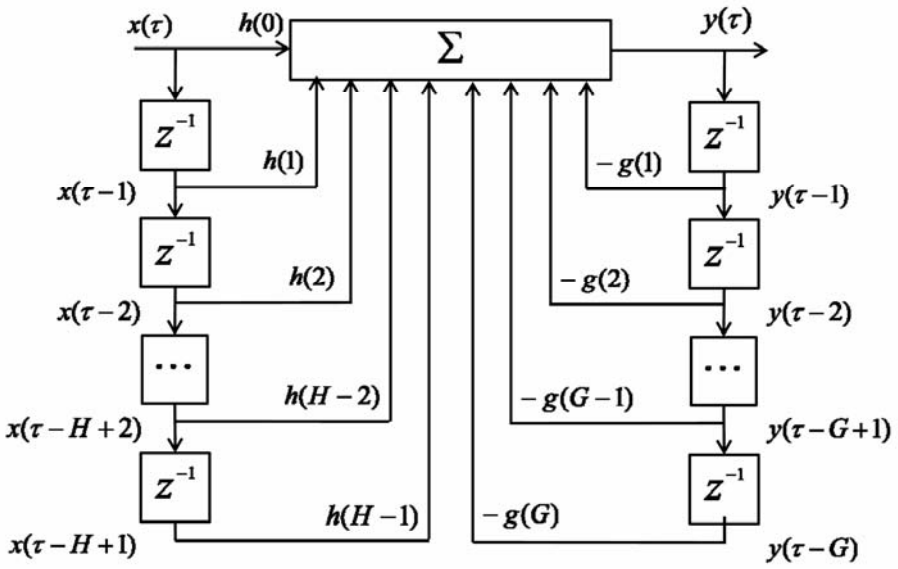


Рисунок 2.17 – Рекурсивный фильтр прямой реализации

Поставим задачу упросить структуру рекурсивного фильтра, уменьшив число блоков задержки. Для этого сведем рекурсивную фильтрацию сигнала $x(\tau)$ к нерекурсивной фильтрации некоторого сигнала $d(\tau)$, построенного на основе сигнала $x(\tau)$,

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times d(\tau - i) \quad (\tau = \overline{0, N-1}). \tag{2.7}$$

Можно показать, что таким сигналом будет сигнал

$$d(\tau) = x(\tau) - \sum_{j=1}^{G} g(j) \times d(\tau - j) \quad (\tau = \overline{0, N-1}). \tag{2.8}$$

Действительно, подставив формулу (2.8) в (2.7) для всех $\tau = \overline{0, N-1}$ имеем

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times \left(x(\tau - i) - \sum_{j=1}^{G} g(j) \times d(\tau - i - j) \right),$$

а после очевидных тождественных преобразований получим

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times x(\tau - i) - \sum_{j=1}^{G} g(j) \times \sum_{i=0}^{H-1} h(i) \times d(\tau - i - j). \tag{2.9}$$

Из (2.7) видно, что последняя сумма в формуле (2.9) не что иное как $y(\tau - j)$. В итоге получаем исходную формулу (2.1), описывающую рекурсивный фильтр в общем виде.

Тогда из (2.7) и (2.8) следует, что рекурсивная фильтрация сигнала $x(\tau)$ может быть осуществлена фильтром, показанным на рисунке 2.18.[39]

[39] Следует заметить, что шумовые характеристики рекурсивного фильтра в канонической форме хуже аналогичных характеристик рекурсивного фильтра при прямой реализации.

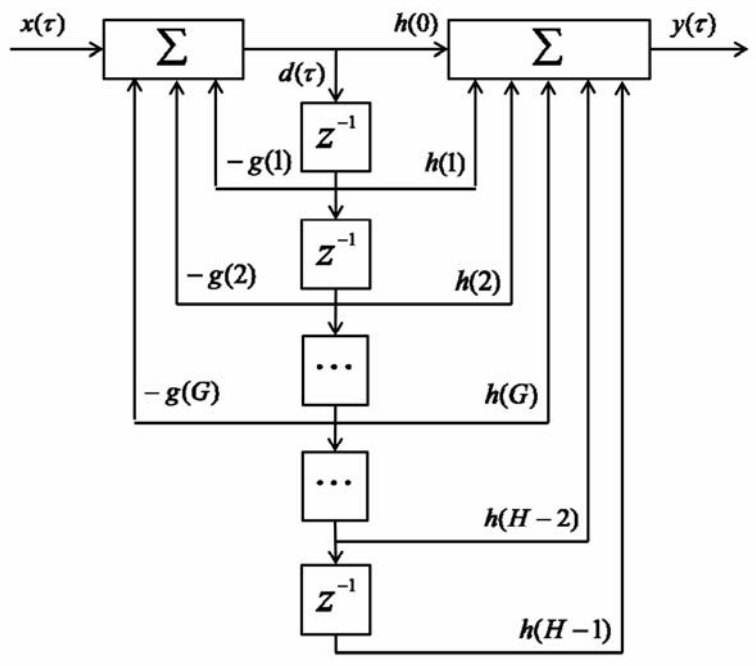


Рисунок 2.18 – Рекурсивный фильтр в канонической форме при условии $G < H$

Основной трудностью, возникающей при проектировании рекурсивных фильтров, является необходимость доказывать его устойчивость. При большом числе коэффициентов эта задача особенно сложна. По этой причине сложные рекурсивные фильтры строятся путем последовательно соединения (каскадирования) более простых фильтров, устойчивость которых легко проверить.

Наиболее простым является биквадратный рекурсивный фильтр (рисунок 2.19), описываемый формулой

$$y(\tau) = h(0) \times x(\tau) + h(1) \times x(\tau - 1) + h(2) \times x(\tau - 2) - g(1) \times y(\tau - 1) - g(2) \times y(\tau - 2).$$

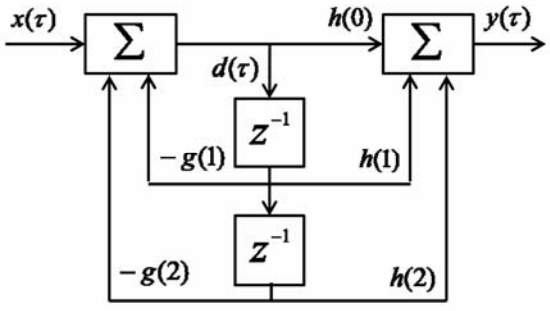


Рисунок 2.19 – Биквадратный рекурсивный фильтр

Пример программы, реализующей каскадный рекурсивный цифровой фильтр, приведен на листинге 2.2. Предполагается, что входные и выходные отсчеты сигнала, а также коэффициенты фильтра хранятся в памяти в формате Q1.15, а коэффициенты

каскадных фильтров упорядочены следующим образом: g1(1), g1(2), h1(0), h1(1), h1(2),…, где g1(0) и g1(2) – коэффициенты $g(1)$ и $g(2)$ первого фильтра, h1(0), h1(1), и h1(2) – коэффициенты $h(0)$, $h(1)$ и $h(2)$ первого фильтра, и так далее.

Листинг 2.2 – Каскадная рекурсивная фильтрация

```
; Исходные данные
.ASG     #010100h, X       ; Адрес входного массива
.ASG     #020200h, Y       ; Адрес выходного массива
.ASG     #030300h, C       ; Адрес массива коэффициентов
.ASG     #040400h, D       ; Адрес буфера задержки
.ASG     4h, L             ; Последний отсчет в буфере задержки
.ASG     32, N             ; Число отсчетов сигнала
.ASG     4, B              ; Число каскадов
; Задание режима работы микпропроцессора
BSET     FRCT, ST1         ; Дробный формат чисел
BSET     SXMD, ST1         ; Знаковый формат операндов
BCLR     M40, ST1          ; 32-разрядные операции
BCLR     SATD, ST1         ; Отключение насыщения при вычислениях
BCLR     SATA, ST3         ; Отключение насыщения при адресации
; Загрузка регистров-указателей
AMOV     #X, XAR0          ; Адрес входного буфера
AMOV     #Y, XAR2          ; Адрес выходного буфера
AMOV     #D, XAR3          ; Адрес буфера задержки
; Настройка циклического буфера задержки
MOV      #2*B, BK03        ; Размер буфера задержки
BSET     AR3LC, ST2        ; Флаг циклической адресации AR3
MOV      mmap(AR3), BSA23  ; Базовый адрес буфера коэффициентов
MOV      #L, AR3           ; Начальное смещение в буфере
; Настройка внешнего и внутреннего циклов
MOV      #(N-1), BRC0      ; Счетчик повторения внешнего цикла
MOV      #(B-1), BRC1      ; Счетчик повторения внутреннего цикла
MOV      #B, T0            ; Число каскадов
; Внешний цикл: t=0, 1, …, N-1
REPLOCAL {
     ; В начало коэффициентов: g1(1),g1(2),h1(0),h1(1),h1(2),…
     AMOV     #C, XAR1
     ; AC0 = x(t)
     MOV      *AR0+ << 16, AC0
     ; Внутренний цикл: i=0, 1, …, B-1
     REPLOCAL {
          ; AC0 -= gi(1)*d(t-1)
          MASM     *(AR3+T0), *AR1+, AC0
          ; AC0 -= gi(2)*d(t-2)
          MASM     T3=*AR3, *AR1+, AC0
          ; Запись d(t) в буфер задержки
          MOV      HI(AC0), *AR3+
          ; AC0 = hi(2)*d(t-2)
          MPYM     *AR1+, T3, AC0
          ; AC0 += hi(0)*d(t)
          MACM     (*AR3+T0), *AR1+, AC0
          ; AC0 += hi(1)*d(t-1)
          MACM     *AR3+, *AR1+, AC0
     ; Запись выходного отсчета
```

Листинг 2.2 – Каскадная рекурсивная фильтрация

```
    MOV      rnd(HI(AC0)), *AR2+
}
```

2.2.4 Адаптивный фильтр

Все способы использования адаптивных фильтров сводятся к решению задачи идентификации, то есть определения характеристик некоторого преобразователя сигнала.

При прямой идентификации адаптивный фильтр включают параллельно с исследуемым преобразователем сигнала (рисунок 2.20). Входной сигнал является общим для исследуемого преобразователя и адаптивного фильтра, а выходной сигнал преобразователя служит для адаптивного фильтра образцовым сигналом. В процессе адаптации временные и частотные характеристики фильтра будут стремиться к соответствующим характеристикам исследуемого преобразователя, а на выходе фильтра входной сигнал будет стремиться к образцовому.

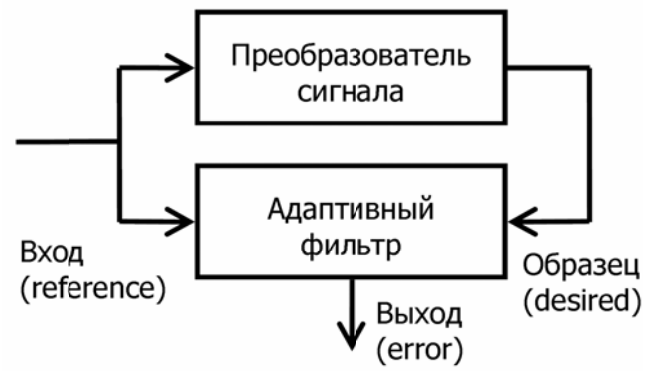


Рисунок 2.20 – Прямая идентификация сигналов

При обратной идентификации адаптивный фильтр включается последовательно с исследуемым преобразователем (рисунок 2.21). Выходной сигнал преобразователя поступает на вход адаптивного фильтра, а входной сигнал преобразователя является образцом для адаптивного фильтра. Фильтр стремится компенсировать на своем выходе влияние преобразователя и восстановить исходный сигнал, устранив внесенные преобразователем искажения.

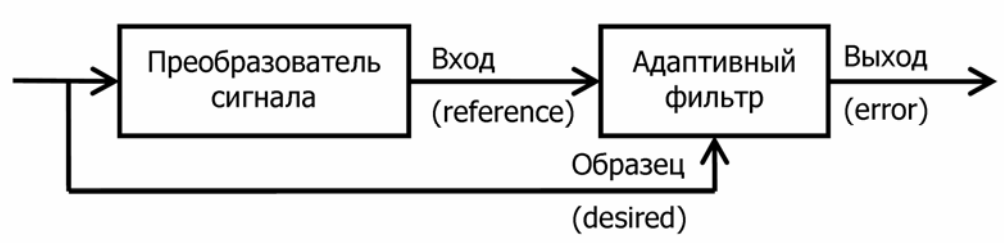


Рисунок 2.21 – Обратная идентификация сигналов

Как при прямой, так и при обратной идентификации адаптивный фильтр имеет два входа: на первый вход фильтра подается сигнал с входа преобразователя, а на второй – с его выхода. Таким образом, адаптивный фильтр располагает сигналами, достаточными для измерений характеристик исследуемого преобразователя, и стремится преобразовать входной сигнал так, чтобы сделать его как можно ближе к образцу.

Рассмотрим пример адаптивной фильтрации. Пусть необходимо реализовать фильтрацию фоновых шумов, появляющихся при использовании микрофона в шумном месте. Для получения шумового сигнала воспользуемся дополнительным микрофоном, который установлен в некотором удалении от основного. Следует заметить, что шум из этого микрофона нельзя просто вычесть из сигнала основного микрофона, поскольку звуковые волны шума претерпевают разные искажения перед фиксацией их микрофонами (рисунок 2.22).

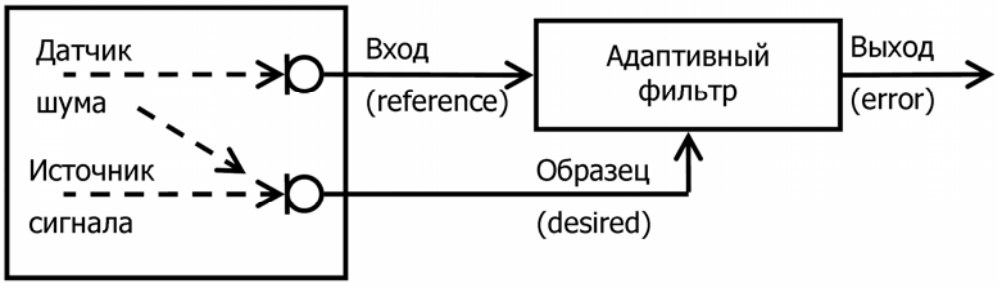


Рисунок 2.22 – Адаптивное подавление шума

Общая структура адаптивного фильтра показана на рисунке 2.23. Адаптивный фильтр состоит из трех компонентов: перестраиваемый цифровой фильтр, блок адаптации коэффициентов и также блок вычитания.

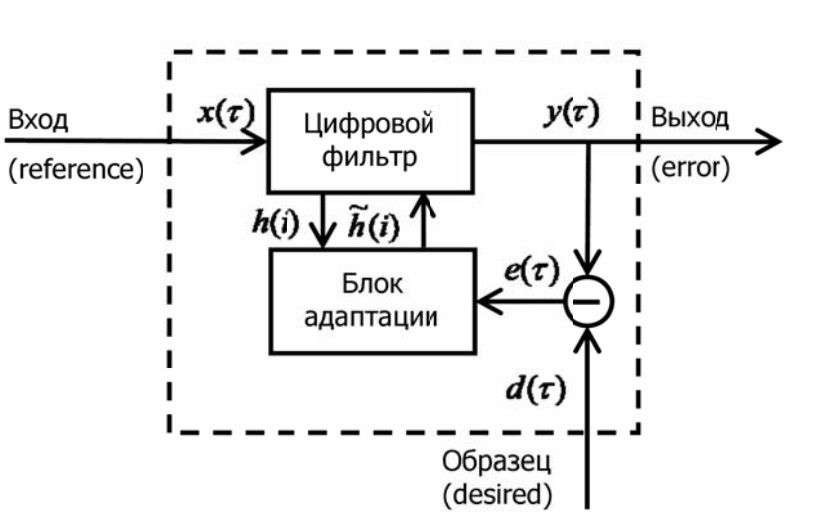


Рисунок 2.23 – Структура адаптивного фильтра

Отсчеты опорного шумового сигнала $x(\tau)$ обрабатываются фильтром, в результате чего получаются отсчеты выходного сигнала $y(\tau)$. Отсчеты выходного сигнала сравниваются с отсчетами образцового сигнала $d(\tau)$, а разность между ними образуют отсчеты сигнала ошибки $e(\tau)$. Задача адаптивного фильтра – минимизировать ошибку воспроизведения образцового сигнала. С этой целью блок адаптации после обработки каждого отсчета сигнала анализирует ошибку и текущие коэффициенты, поступающие из фильтра, и использует результаты этого анализа для подстройки коэффициентов фильтра.

Блок адаптации вносит в процесс фильтрации обратную связь, вследствие чего адаптивный фильтр может оказаться неустойчивым. Одним из достоинств нерекурсивного фильтра является его устойчивость при любых значениях коэффициентов. Поэтому в качестве цифрового фильтра при адаптивной фильтрации чаще всего используют нерекурсивный фильтр

$$y(\tau) = \sum_{i=0}^{H-1} h(i) \times x(\tau - i) \quad (\tau = \overline{0, N-1}),$$

где N – число отсчетов входного $x(\tau)$ и выходного сигналов $y(\tau)$, $h(i)$ – импульсная характеристика фильтра.

Сигнал ошибки $e(\tau)$ вычисляется как разность отсчетов образцового сигнала $d(\tau)$ и выходного сигнала $y(\tau)$,

$$e(\tau) = d(\tau) - y(\tau).$$

Блок адаптации при вычислении нового отчета выходного сигнала $y(\tau+1)$ по завершению каждого шага i корректирует текущий коэффициент $h(i)$,

$$\widetilde{h}(i) = h(i) + \mu \times e(\tau) \times x(\tau - i) \quad (i = \overline{0, H-1}),$$

где $\widetilde{h}(i)$ – новое значение коэффициента $h(i)$, μ – константа адаптации (конверсии) коэффициентов фильтра, $e(\tau)$ – текущая ошибка, вычисленная для предыдущего выходного отсчета $y(\tau)$, $x(\tau - i)$ – текущий входной отсчет.

Для реализации адаптивной фильтрации сигналов используется специальная команда микропроцессора LMS, имеющая 4 операнда:

– Xmem – текущее значение коэффициента, адресуемое через дополнительные регистры-указатели;

– Ymem – текущее значение входного отсчета, адресуемое через дополнительные регистры-указатели;

– ACx – регистр-аккумулятор AC0-AC3, содержащий обновленный коэффициент;

– ACy – регистр-аккумулятор AC0-AC3, накапливающий значение выходного отсчета.

Команда выполняет параллельные операции умножения с накоплением и сложения с округлением:

ACy = ACy + (Xmem * Ymem) – сложение с ACy результата умножения текущего коэффициента фильтра на текущий входной отсчет;

ACx = rnd(ACx + (Xmem << #16)) – сложение текущего коэффициента фильтра с корректирующим значением в ACx и округление полученного значения.

Пример программы, реализующей адаптивный цифровой фильтр, приведен на листинге 2.3, где в отличие от предыдущей реализации нерекурсивного фильтра используется буфер задержанного сигнала.

Листинг 2.3 – Адаптивная фильтрация

```
; Исходные данные
.ASG     010100h, X        ; Адрес входного сигнала
.ASG     020200h, Y        ; Адрес выходного сигнала
.ASG     030300h, D        ; Адрес образцового сигнала
.ASG     040400h, C        ; Адрес коэффициентов фильтра
.ASG     050500h, Z        ; Адрес буфера задержки
.ASG     4h, L             ; Последний отсчет в буфере задержки
.ASG     32, N             ; Число отсчетов сигнала
.ASG     8, H              ; Число коэффициентов фильтра
.ASG     1000h, u          ; Константа адаптации 0,125
```

Листинг 2.3 – Адаптивная фильтрация

```
.ASG     0h, ue             ; Начальная ошибка ue
; Задание режима работы микпропроцессора
BSET     FRCT, ST1          ; Дробный формат чисел
BSET     SXMD, ST1          ; Знаковый формат операндов
BCLR     M40, ST1           ; 32-разрядные операции
BCLR     SATD, ST1          ; Отключение насыщения при вычислениях
BCLR     SST, ST3           ; Отключение насыщения при сохранении
; Загрузка регистров-указателей
AMOV     #X, XAR0           ; Адрес входного буфера
AMOV     #C, XAR1           ; Адрес буфера коэффициентов
AMOV     #Y, XAR2           ; Адрес выходного буфера
AMOV     #D, XAR3           ; Адрес образцового буфера
AMOV     #Z, XAR4           ; Адрес буфера задержки
MOV      #u, T0             ; Константа адаптации
; Настройка циклического буфера коэффициентов
MOV      #H, BK03           ; Размер буфера коэффициентов
BSET     AR1LC, ST2         ; Флаг циклической адресации для AR1
MOV      mmap(AR1), BSA01 ; Базовый адрес буфера коэффициентов
MOV      #0, AR1            ; Начальное смещение коэффициентов
; Настройка циклического буфера задержки
MOV      #H, BK47           ; Размер буфера задержки
BSET     AR4CL, ST2         ; Флаг циклической адресации для AR4
MOV      mmap(AR4), BSA45 ; Базовый адрес буфера задержки
MOV      #L, AR4            ; Смещение последнего отсчета
; Настройка внешнего и внутреннего циклов
MOV      #(N-1), BRC0       ; Счетчик повторения внешнего цикла
MOV      #(H-1), BRC1       ; Счетчик повторения внутреннего цикла
; Начальный корректирующий коэффициент u*e(0)
MOV      #ue, T3
; Внешний цикл: t=0, 1, …, N-1
REPLOCAL {
     ; Запись нового отсчета x(t) в буфер задержки
     MOV      *AR0+,*+AR4
     ; Обнуление выходного отсчета y(t)
     MOV #0, AC1
     ; Внутренний цикл: i=0, 1, …, H-1
     REPLOCAL {
          ; Вычисление корректирующего значения u*e(t)*x(t-i)
          MPYM     *AR4, T3, AC0
          ; Вычисление AC1+=h(i)*x(t-i) || AC0+=h(i)
          LMS      *AR1, *AR4-, AC0, AC1
          ; Сохранить скорректированный коэффициент h(i)
          MOV      HI(AC0), *AR1+
     }
     ; Запись выходного отсчета y(t)
     MOV      rnd(HI(AC1)), *AR2+
     ; Вычисление ошибки e(t)
     SUB      AC1, *AR3+ << #16, AC2
     ; Вычисление корректирующего коэффициента u*e(t)
     MPYR     T0, AC2, AC1
     ; Сохранение корректирующего коэффициента u*e(t) в T3
     MOV      HI(AC1), T3
}
```

2.2.5 Дискретное преобразование Фурье

Дискретное преобразование Фурье (ДПФ) сигнала $x(\tau)$ вычисляется по формуле

$$c(i) = \sum_{\tau=0}^{N-1} x(\tau) \times \omega_N^i(\tau) \quad (i = \overline{0, N-1}), \tag{2.10}$$

где $c(i)$ – спектр сигнала, $\omega_N^i(\tau)$ – комплексные спектральные функции,

$$\omega_N^i(\tau) = \cos(2\pi i \tau / N) - j \sin(2\pi i \tau / N) = \exp(-2\pi j i \tau / N), \tag{2.11}$$

j – комплексная единица. Если N четно, то (2.10) можно переписать следующим образом,

$$c(i) = \sum_{\tau=0}^{N/2-1} x(2\tau) \times \omega_N^i(2\tau) + \sum_{\tau=0}^{N/2-1} x(2\tau+1) \times \omega_N^i(2\tau+1),$$

а с учетом (2.11),

$$c(i) = \sum_{\tau=0}^{M-1} x(2\tau) \times \omega_M^i(\tau) + w_N^i \times \sum_{\tau=0}^{M-1} x(2\tau+1) \times \omega_M^i(\tau), \tag{2.12}$$

где $M = N/2$, w_N^i – поворачивающий множитель, не зависящий от номера отсчета входного сигнала τ,

$$w_N^i = \omega_N^i(1) = \exp(-2\pi j i / N).$$

В итоге дискретное преобразование (2.10) сигнала $x(\tau)$ с N отсчетами и требующее N^2 операций комплексного умножения со сложением преобразовано в линейную комбинацию двух дискретных преобразований Фурье половин сигнала $x(2\tau)$ и $x(2\tau+1)$ с M отсчетами, для чего необходимо выполнить всего лишь $N^2/2+1$ операций комплексного умножения со сложением.

Если N является степенью двойки, то деление входного сигнала можно продолжать до тех пор, пока не будет получено двухточечное преобразование:

$$\begin{cases} c(0) = x(0) + w_2^0 \times x(1) = x(0) + x(1); \\ c(1) = x(0) + w_2^1 \times x(1) = x(0) - x(1). \end{cases}$$

Таким образом, быстрое преобразование Фурье (БПФ) – это алгоритм эффективного вычисления дискретного преобразования Фурье с числом комплексных операций умножения со сложением порядка $N \log N$. Диаграммы вычисления быстрого преобразования Фурье при различных значениях N представлены на рисунке 2.24.

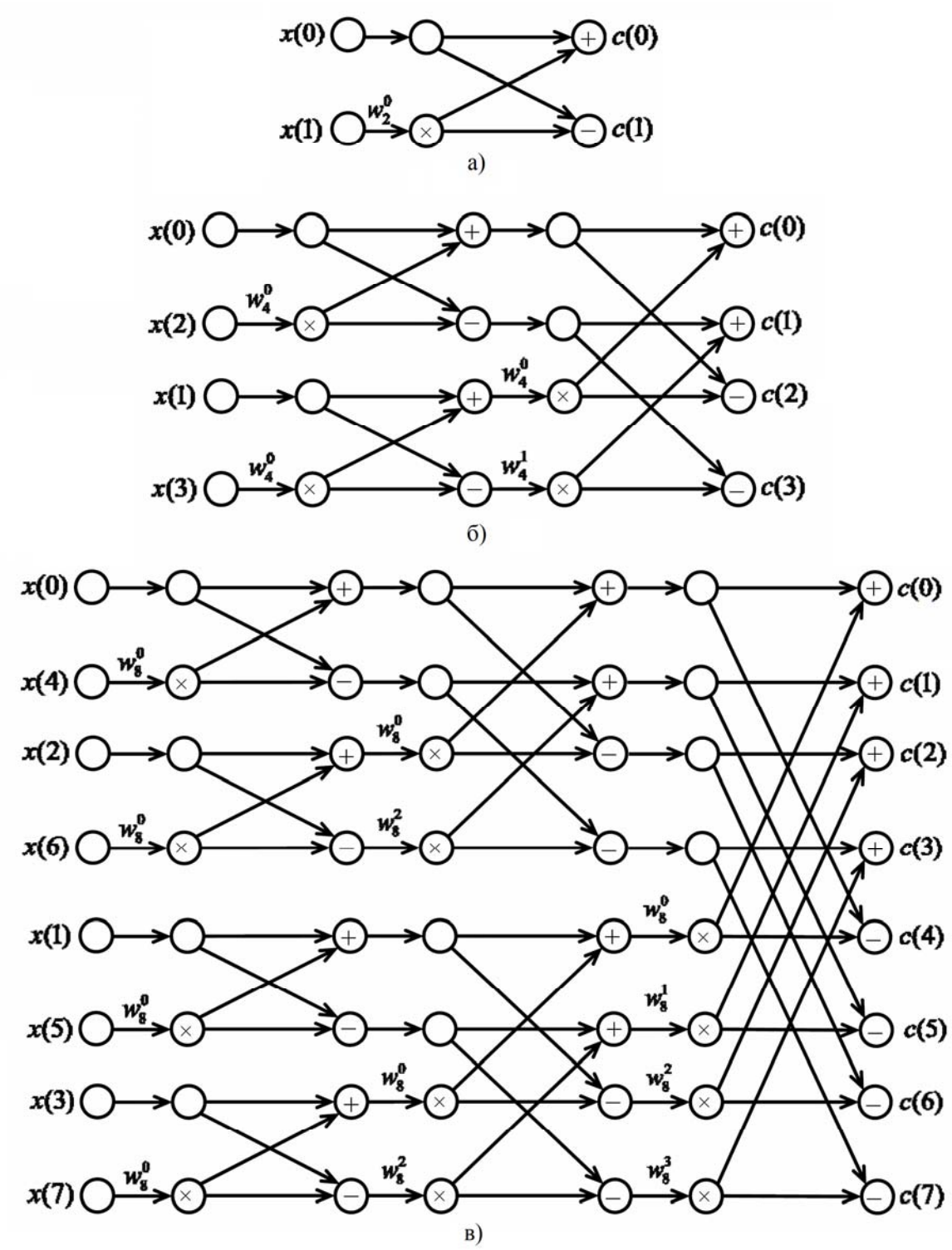


Рисунок 2.24 – Диаграммы БПФ: а) $N=2$, б) $N=4$, в) $N=8$

Из диаграмм видно, что отсчеты сигнала, подаваемые на вход БПФ, размещаются в бит-реверсивном порядке, например, при $N=8$ входной отсчет 000 – это отсчет 000 сигнала, 001 – 100, 010 – 010, 011 – 110, 100 – 001, 101 – 101, 110 – 011, 111 – 111.

Обратное преобразование Фурье осуществляется аналогично прямому:

$$y(\tau) = \sum_{i=0}^{N-1} c(i) \times \omega_N^{\tau}(i) \quad (\tau = \overline{0, N-1}).$$

Аппаратурный ускоритель микропроцессора (англ. hardware accelerator of fast Fourier transform, HWAFFT) реализует эффективное выполнение быстрого преобразования Фурье [33]. Конструктивно аппаратурный ускоритель размещен вне ядра микропроцессора, но сильно связан с ним. По отношению к ядру микропроцессора аппаратурный ускоритель является сопроцессором: в процессе функционирования аппаратурный ускоритель просматривает команды, поступающие в ядро микропроцессора, выбирает ему предназначенные команды, и во время их выполнения использует ресурсы ядра. Аппаратурный ускоритель подключен к магистралям BB, CB и DB по чтению, имеет доступ к регистрам ядра микропроцессора и управляет работой генератора адресов данных. Так как аппаратурный ускоритель выступает в роли исполнительного устройства и завершает свою работу на исполнительной стадии конвейера, сохранение данных в памяти осуществляет ядро микропроцессора на стадии записи результата команды.

В постоянном запоминающем устройстве микропроцессора хранится исполняемый код функций прямого (англ. fast Fourier transform, FFT) и обратного (англ. inverse fast Fourier transform, IFFT) преобразования Фурье (таблица 2.1). Для ускорения преобразования таблица с 512 комплексными значениями поворачивающих множителей w_{1024}^{i} ($i = \overline{0, 511}$) размещена внутри самого аппаратурного ускорителя.[40]

Таблица 2.1 – Встроенные функции аппаратурного ускорителя

Адрес	Прототип
FF6CD6h	void hwafft_br(long *in, long *out, unsigned len);
FF6CEAh	unsigned hwafft_8pts(long *in, long *out, unsigned fft, unsigned scale);
FF6DD9h	unsigned hwafft_16pts(long *in, long *out, unsigned fft, unsigned scale);
FF6F2Fh	unsigned hwafft_32pts(long *in, long *out, unsigned fft, unsigned scale);
FF7238h	unsigned hwafft_64pts(long *in, long *out, unsigned fft, unsigned scale);
FF73CDh	unsigned hwafft_128pts(long *in, long *out, unsigned fft, unsigned scale);
FF75DEh	unsigned hwafft_256pts(long *in, long *out, unsigned fft, unsigned scale);
FF77DCh	unsigned hwafft_512pts(long *in, long *out, unsigned fft, unsigned scale);
FF7A56h	unsigned hwafft_1024pts(long *in, long *out, unsigned fft, unsigned scale);

[40] При N=1024 хранится только половина поворачивающих множителей, т.к. $w_N^{i+N/2} = -w_N^{i}$, а для других значений N используется тождество $w_{N/2}^{i} = w_N^{2i}$.

Входные и выходные данные встроенных функций – массивы комплексных чисел, каждое из которых является двойным словом и состоит из двух дробных чисел в формате Q1.15 – действительной (старшее слово) и мнимой части (младшее слово).

Встроенные функции аппаратурного ускорителя hwafft_Npts, где N – длина массива входных и выходных данных, позволяют вычислять прямое или обратное быстрое преобразование Фурье в зависимости от состояния флага fft: 0 – выполняется прямое преобразование, 1 – обратное.

Для ускорения доступа к данным входной и выходной массив рекомендуется размещать в различных блоках встроенной памяти микропроцессора. Расположение результата преобразования является переменным и зависит от возвращаемого значения функции. Если функция возвращает ноль, то результат записан во входной буфер in, а если не ноль, результат помещен в выходной буфер out.

Флаг scale необходим для масштабирования результата преобразования: 0 – после каждого этапа преобразования результат делится на два, 1 – масштабирование не производится. В результате масштабирования реализуется нормированное прямое и обратное дискретное преобразование Фурье:

$$c(i) = \frac{1}{N}\sum_{\tau=0}^{N-1} x(\tau) \times \omega_N^i(\tau) \quad (i = \overline{0, N-1});$$

$$y(\tau) = \frac{1}{N}\sum_{i=0}^{N-1} c(i) \times \omega_N^\tau(i) \quad (\tau = \overline{0, N-1}).$$

В связи с тем, что вычисление прямого и обратного преобразования Фурье требует бит-реверсивного порядка элементов во входном массиве in (см. рисунок 2.24), реализована также функция hwafft_br, выполняющая ускоренную бит-реверсивную перестановку элементов в массивах комплексных чисел. Для ускорения бит-реверсивной перестановки элементов рекомендуется размещать выходной массив данных out с выравниванием по границе области памяти с размером $4N$, т.е. адрес массива out должен содержать $\log_2(4N)$ нулевых младших бит.

Пример реализации на языке С спектральной обработки сигнала приведен на листинге 2.4.

Листинг 2.4 – Спектральная обработка сигнала

```
#define FFT           (0)          /* Прямое преобразование */
#define IFFT          (1)          /* Обратное преобразование */
#define SCALE         (0)          /* Флаг масштабирования */
#define NOSCALE       (1)          /* Отсутствие масштабирования */
#define SEL_IN        (0)          /* Результат во входном массиве */
#define SEL_OUT       (1)          /* Результат в выходном массиве */
#define HWAFFT_BR     (0xff6cd6)   /* Адрес функции hwafft_br */
#define HWAFFT_1024   (0xff7a56)   /* Адрес функции hwafft_1024pts */
/* Объявление внешних функций чтения-записи отсчетов сигнала */
    extern long get(unsigned ind);
    extern void put(unsigned ind, long);
/* Объявление внешней функции преобразования спектра */
    extern void fun(long *inp, long *out, unsigned len);
/* Прототипы встроенных функций аппаратурного ускорителя */
    void (*hwafft_br)( long*, long*, unsigned);
    unsigned (*hwafft_1024pts)( long*, long*, unsigned, unsigned);
/* Объявление массивов входных и выходных данных */
    long onchip inp[1024];
    long onchip out[1024];
/* Объявление массива для бит-реверсивной перестановки элементов */
#   pragma DATA_ALIGN(rev, 4096)
    long onchip rev[1024];
/* Объявление временных переменных */
    unsigned ind, sel;
    long *ptr;
/* Получение комплексных отсчетов входного сигнала */
#   pragma UNROLL(4)
    for(ind = 0, ptr = inp; ind < 1024; ind++, ptr++)
        *ptr = get(ind);
/* Бит-реверсивная перестановка элементов входного массива */
    (*(hwafft_br)HWAFFT_BR)(inp, rev, 1024);
/* Прямое преобразование Фурье */
    sel = (*(hwafft_1024pts)HWAFFT_1024)(rev, out, FFT, SCALE);
    if(sel == SEL_IN )
        ptr = rev;
    else
        ptr = out;
/* Преобразование спектра сигнала */
    fun(ptr, inp, 1024);
/* Бит-реверсивная перестановка элементов спектра */
    (*(hwafft_br)HWAFFT_BR)(inp, rev, 1024);
/* Обратное преобразование Фурье */
    sel = (*(hwafft_1024pts)HWAFFT_1024)(rev, out, IFFT, SCALE);
    if(sel == SEL_IN )
        ptr = rev;
    else
        ptr = out;
/* Выдача комплексных отсчетов выходного сигнала */
#   pragma UNROLL(4)
    for(ind = 0; ind < 1024; ind++, ptr++)
        put(i, *ptr);
```

В таблице 2.2 приведены результаты измерения числа циклов микропроцессора и потребленной энергии, затраченных на выполнение быстрого преобразования Фурье с использованием аппаратурного ускорителя и оптимизированной программы для микропроцессора.

Таблица 2.2 – Эффективность аппаратурного ускорителя (1,05 В, 60 МГц) [33]

N	Ускоритель, циклы[41] (энергия в нДж)	Программа, циклы (энергия в нДж)	Эффективность, в разах
8	92 + 38 = 130 (23,6)	196 + 95 = 291 (95,1)	2,2 (4,0)
16	115 + 55 = 170 (32,1)	344 + 117 = 461 (157,1)	2,7 (4,9)
32	234 + 87 = 321 (69,5)	609 + 139 = 748 (269,9)	2,3 (3,9)
64	285 + 151 = 436 (98,5)	1194 + 211 = 1405 (531,7)	3,2 (5,4)
128	633 + 279 = 912 (219,2)	2499 + 299 = 2798 (1090,4)	3,1 (5,0)
256	1133 + 535 = 1668 (407,2)	5404 + 543 = 5947 (2354,2)	3,6 (5,8)
512	2693 + 1047 = 3740 (939,7)	11829 + 907 = 12736 (5097,5)	3,4 (5,4)
1024	5244 + 2071 = 7315 (1836,2)	25934 + 1783 = 27717 (11097,9)	3,8 (6,0)

Выполнение прямого быстрого преобразования Фурье при большем числе отсчетов осуществляется на основе формулы (2.12), из которой следует:

$$c(i) = c_1(i) + w_N^i \times c_2(i) \quad (i = \overline{0, N/2-1});$$

$$c(N/2+i) = c_1(i) - w_N^i \times c_2(i) \quad (i = \overline{0, N/2-1}),$$

где $c_1(i)$ и $c_2(i)$ – спектры, полученные при прямом преобразовании Фурье четных и нечетных отсчетов входного сигнала. Аналогичные формулы справедливы и для обратного быстрого преобразования Фурье:

$$y(\tau) = y_1(\tau) + w_N^\tau \times y_2(\tau) \quad (\tau = \overline{0, N/2-1});$$

$$y(N/2+\tau) = y_1(\tau) - w_N^\tau \times y_2(\tau) \quad (\tau = \overline{0, N/2-1}),$$

где $y_1(\tau)$ и $y_2(\tau)$ – сигналы, полученные при обратном преобразовании Фурье четных и нечетных отсчетов спектра.

[41] Число циклов, необходимых для выполнения быстрого преобразования Фурье, складывается из двух частей: числа циклов, необходимых для бит-реверсивной перестановки элементов массива, и числа циклов, затраченных на дискретное преобразование.

2.3 Домашнее задание 2

2.3.1 Общие указания

Целью выполнения домашнего задания является изучение алгоритмов обработки сигналов и данных, заданных в теме домашнего задания. Темы домашних заданий приведены в пп. 2.3.2.

Для выполнения домашнего задания предварительно необходимо ознакомиться с общим описанием библиотеки цифровой обработки сигналов [47], найти и изучить описание интерфейса библиотечной функции, заданной в теме домашнего задания.

На следующем этапе выполнения домашнего задания необходимо в литературе, например [1, 14, 15, 16, 18, 20, 21, 24], найти и привести в отчете те теоретические сведения, которые необходимы для понимания метода решения задачи индивидуального задания. Далее в отчете описывается алгоритм, реализуемый библиотечной функцией.

На последнем этапе выполнения домашнего задания необходимо изучить исходный текст заданной библиотечной функции, прилагаемый к описанию библиотеки цифровой обработки сигналов. Результатом изучения исходного текста должны быть комментарии к операторам программы, содержащие ссылки на теоретический материал и указания на используемые технические средства микропроцессора. Прокомментированный текст библиотечной функции приводится в приложении к отчету по домашнему заданию.

Итогом выполнения домашнего задания является оформление отчета. Срок выполнения домашнего задания – две недели с момента выдачи темы индивидуального задания.

2.3.2 Индивидуальные задания

Ниже приведены темы индивидуальных заданий, заключающиеся в исследовании следующих функций библиотеки цифровой обработки сигналов [47]:

1) вычисления автокорреляционной функции acorr [47, с. 28];
2) сложения двух векторов add [47, с. 30];
3) вычисления арктангенсов atan16 [47, с. 32];
4) вычисления минимального порядка элементов вектора bexp [47, с. 34];
5) вычисления коротких бит-реверсивных индексов cbrev [47, с. 35];

6) вычисления длинных бит-реверсивных индексов cbrev32 [47, с. 36];
7) прямого комплексного быстрого преобразование Фурье cfft [47, с. 37];
8) прямого комплексного быстрого преобразование Фурье cfft32 [47, с. 40];
9) комплексной рекурсивной фильтрации cfir [47, с. 42];
10) обратного комплексного быстрого преобразование Фурье cifft [47, с. 47];
11) обратного комплексного быстрого преобразование Фурье cifft32 [47, с. 49];
12) обычной свертки двух векторов convol [47, с. 52];
13) ускоренной свертки двух векторов convol1 [47, с. 54];
14) быстрой свертки двух векторов convol2 [47, с. 56];
15) вычисления корреляционной функции corr [47, с. 58];
16) обычной адаптивной среднеквадратической фильтрации dlms [47, с. 60];
17) быстрой адаптивной среднеквадратической фильтрации dlmsfast [47, с. 62];
18) возведения вектора в натуральную степень expn [47, с. 66];
19) фильтрации с конечной импульсной характеристикой fir [47, с. 67];
20) фильтрации с конечной импульсной характеристикой fir2 [47, с. 70];
21) децимационной нерекурсивной фильтрации firdec [47, с. 73];
22) интерполяционной нерекурсивной фильтрации firinterp [47, с. 75];
23) прямой сетчатой нерекурсивной фильтрация firlat [47, с. 77];
24) симметричной нерекурсивной фильтрация firs [47, с. 79];
25) преобразования чисел с плавающей запятой в формат Q15 fltq15 [47, с. 82];
26) преобразования Гильберта нерекурсивным фильтром hilb16 [47, с. 83];
27) рекурсивной фильтрация двойной точности iir32 [47, с. 87];
28) каскадной рекурсивной фильтрации с 4 коэффициентами iircas4 [47, с. 89];
29) каскадной рекурсивной фильтрации с 5 коэффициентами iircas5 [47, с. 91];
30) каскадной рекурсивной фильтрации с 5 коэффициентами iircas51 [47, с. 93];
31) инверсной сетчатой рекурсивной фильтрация iirlat [47, с. 95];
32) поэлементного деления векторов ldiv16 [47, с. 97];
33) вычисления десятичного логарифма элементов вектора log_10 [47, с. 98];
34) вычисления двоичного логарифма элементов вектора log_2 [47, с. 100];
35) вычисления натурального логарифма элементов вектора logn [47, с. 101];
36) нахождения индекса максимального элемента вектора maxidx [47, с. 102];
37) нахождения индекса максимального элемента вектора maxidx34 [47, с. 104];
38) нахождения максимального значения элементов вектора maxval [47, с. 105];
39) нахождения индекса и значения максимального элемента maxvec [47, с. 105];

40) нахождения индекса минимального элемента вектора minidx [47, с. 106];
41) нахождения минимального значения элементов вектора minval [47, с. 107];
42) нахождения индекса и значения минимального элемента minvec [47, с. 108];
43) умножения матриц mmul [47, с. 109];
44) транспонирования матриц mtrans [47, с. 110];
45) умножения векторов двойной точности mul32 [47, с. 111];
46) изменения знака элементов вектора neg [47, с. 112];
47) изменения знака элементов вектора двойной точности neg32 [47, с. 112];
48) возведения в квадрат элементов вектора power [47, с. 114];
49) преобразования вектора в формат с плавающей запятой q15tofl [47, с. 115];
50) генерации псевдослучайного вектора rand16 и rand16init [47, с. 116];
51) вычисления обратных значений элементов вектора recip16 [47, с. 119];
52) прямого действительного преобразование Фурье rfft| [47, с. 123];
53) прямого действительного преобразование Фурье rfft32 [47, с. 123];
54) обратного действительного преобразование Фурье rifft [47, с. 124];
55) обратного действительного преобразование Фурье rifft32 [47, с. 126];
56) вычисления синусов элементов вектора sine [47, с. 127];
57) вычисления квадратного корня элементов вектора sqr_16 [47, с. 128];
58) вычитания векторов sub [47, с. 129].

2.3.3 Пример выполнения домашнего задания 2

Оформление отчета по домашнему заданию осуществляется в соответствии с ГОСТ 2.105-95 [8], список литературы – по ГОСТ 7.1-2003 [9]. Отчет должен содержать:

- титульный лист (приложение В);
- содержание;
- постановка задачи;
- теоретические сведения;
- интерфейс библиотечной функции;
- алгоритм обработки сигналов;
- список литературы;
- приложение с текстом библиотечной функции.

Ниже приведен пример выполнения и оформления домашнего задания на тему «Симметричное канальное шифрование сигналов».

2.3.3.1 Постановка задачи

Темой домашнего задания является изучение алгоритмов и исследование способов их реализации, выполняющих канальное шифрование сигналов в цифровой форме. Для выполнения домашнего задания необходимо:

- ознакомиться с рекомендованной литературой;
- привести в отчете теоретические сведения по методам обработки сигналов;
- описать алгоритм обработки сигналов и данных;
- прокомментировать текст библиотечной функции;
- оформить отчет по домашнему заданию.

Основным результатом домашнего задания является комментированный текст библиотечной функции на языке ассемблера, а также описание выполняемого ею алгоритма цифровой обработки сигналов и данных.

2.3.3.2 Канальное шифрование сигналов

Канальное шифрование – способ шифрования, при котором шифруются данные, проходящие через канал связи, а также данные о маршрутизации сообщений и форматах их представления. Канальное шифрование применяется при передаче конфиденциальных данных по незащищенным каналам связи. При шифровании выполняется обратимое преобразование данных с целью их сокрытия от неавторизованных получателей и, в это же время, предоставление авторизованным получателям возможности доступа к этим данным в незашифрованном виде.

Шифром называется пара алгоритмов, реализующих прямое и обратное преобразование сообщений. Эти алгоритмы применяются над данными с использованием ключа, который задает выбор конкретного преобразования из совокупности возможных преобразований данных, реализуемых заданным алгоритмом.

Ключи для шифрования и для расшифровывания могут отличаться, а могут быть одинаковыми. В симметричных криптосистемах для шифрования и расшифрования используется один и тот же ключ. Алгоритм и ключ выбирается заранее и известен обеим сторонам.

Основной операцией симметричного шифрования является гаммирование, или поразрядное сложение по модулю два криптографической гаммы с блоком исходных данных, которое используется для получения шифротекста. Криптографическая гамма вычисляется по некоторому известному алгоритму на основе ключа (см. ГОСТ 28147-89).

2.3.3.3 Интерфейс библиотечной функции

Для выполнения операции гаммирования используется функция gamma, имеющая следующий прототип на языке С:

void gamma(DATA *x, DATA *y, DATA *r, ushort nx),

где ushort – тип данных, определенный как unsigned short; DATA – тип данных, являющийся синонимом int; x – указатель на вектор исходных данных; y – указатель на вектор криптографической гаммы, r – указатель на вектор, выделенный вызывающей функцией для размещения результатов гаммирования; nx – число элементов в векторах x, y и z. Возвращаемое значение y функции отсутствует.

Векторы входных и выходных данных функции представляют собой одномерные массивы x, y и z, элементами которых являются 16-разрядные числа (целые или дробные), пронумерованные в порядке увеличения их адресов в памяти данных (рисунок 2.25).

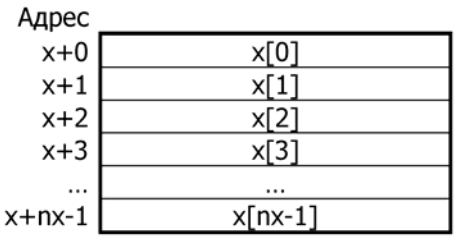


Рисунок 2.25 – Вектор x функции gamma

Длина векторов nx задается в формате N16 – беззнаковом 16-разрядном целочисленном формате.

Сами массивы, а также их элементы выравнены по границе слова. Разрешается использование одного и того же указателя для задания любых двух векторов. Использование одного и того же указателя для всех трех векторов приведет к обнулению заданного вектора.

Перекрытие массивов не допускается, так как результат вызова функции становится не соответствующий ее спецификации.

2.3.3.4 Алгоритм гаммирования

Функция gamma выполняет гаммирование двух входных векторов x и y, а результат записывает в выходной вектор r. Реализуемый функций gamma алгоритм описан на языке программирования С и приведен на листинге 2.5.

Листинг 2.5 – Алгоритм гаммирования

```
#define  DATA     int
#define  ushort   unsigned short
void gamma(DATA *x, DATA *y, DATA *r, ushort nx)
{
     ushort i;
     for(i = 0; i < nx; i++)
          r[i] = x[i] ^ y[i];
}
```

В теле основного цикла функции выполняется извлечение очередных элементов векторов x и y, их поразрядное сложение по модулю 2 и запись результата в соответствующий элемент выходного вектора. Для организации цикла используется целочисленная переменная i, которая задает индекс элементов векторов.

2.3.3.5 Библиотечная функция

Комментированный текст библиотечной функции gamma приведен в приложении F.

2.4 Лабораторная работа 2

2.4.1 Общие указания

Целью работы является изучение стандартных процедур обработки сигналов и данных, а также их реализация в интегрированной среде проектирования Code Composer Studio™ версии 5 и на микропроцессоре TMS320C5515™ компании Texas Instruments Incorporated®. При выполнении лабораторной работы используется экспериментальная плата TMS320C5515™ Evaluation Module (EVM) компании Spectrum Digital Incorporated® [64].

Лабораторная работа заключается в разработке программы обработки сигналов на языке программирования C, и ее сравнение по эффективности с аналогичной функцией из состава стандартной библиотеки для обработки сигналов [47]. Темы индивидуальных заданий приведены в п. 2.3.2.

Для подготовки к лабораторной работе 2 необходимо:

– повторить материал лекций по обработке сигналов и данных, а также по программированию микропроцессора TMS320C5515™;

– изучить исходные тексты программ из приложений G и H;

– разработать исходный текст функции на языке С, реализующей алгоритм обработки сигналов и данных, заданный в теме индивидуального задания;

– подготовить исходный текст функции из состава стандартной библиотеки обработки сигналов, реализующей тот же алгоритм обработки данных, для ее ассемблирования и выполнения в среде CCS.

Выполнение лабораторной работы состоит из отладки и профилирования программы, выполняющей обработку сигналов и данных, заданных в теме индивидуального задания. Реализация алгоритма обработки осуществляется двумя средствами: тестовой функцией, написанной на языке С, и функцией из состава стандартной библиотеки обработки сигналов.

Профилирование программы заключается в измерении числа циклов микропроцессора, затраченных на выполнение обработки одних и тех же тестовых данных. Тестовые данные предусмотрены для каждой библиотечной функции и расположены в папке *Examples* стандартной библиотеки для обработки сигналов.

2.4.2 Выполнение лабораторной работы

2.4.2.1 Подключение оценочной платы

2.4.2.1.1 Установите исходное состояние перемычек оценочной платы в соответствии с подразделом **ПI.2** приложения I.

2.4.2.1.2 Подключите выход блока питания (рисунок ПI.2а приложения I) к разъему экспериментальной платы J16, а интерфейсным кабелем USB (рисунок ПI.2б приложения I) соедините разъем J2 оценочной платы с разъемом USB компьютера.

2.4.2.1.3 Подключите блок питания к сети и включите питание платы переключателем SW5.

2.4.2.2 Создание проекта

2.4.2.2.1 В интегрированной среде CCS создайте новый проект (см. пп. 1.4.2.1).

2.4.2.2.2 При настройке проекта в разделе *Device*:

– в списке *Family* выберите семейство микропроцессоров *C5500*;

– в списке *Variant* задайте модель микропроцессора *TMS320C5515*;

– в списке *Connection* установите тип оценочной платы *Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator*.

2.4.2.3 Выбор целевой платформы

2.4.2.3.1 До отладки проекта необходимо выбрать и сконфигурировать целевую платформу – эмулятор оценочной платы. Для выбора целевой платформы создайте

конфигурационный файл, для чего в контекстном меню проекта выберите пункт *New* → *Target Configuration File*.

2.4.2.3.2 В открывшемся окне задайте имя конфигурационного файла, например, *C5515_emulator*, к которому будет добавлено стандартное расширение *.ccxml*. По завершению ввода нажмите кнопку *Finish*.

2.4.2.3.3 В появившейся вкладке с именем созданного конфигурационного файла (*C5515_emulator.ccxml*) выберите в ниспадающем списке *Connection* выберите тип экспериментальной платы *Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator*, а в списке *Board or Device* установите флаг модели микропроцессора *TMS320C5515*.

2.4.2.3.4 Завершите конфигурирование, нажав на кнопку *Save* в окне конфигурационного файла. Проконтролируйте, что конфигурационный файл появился в списке файлов проекта с установленным атрибутом активности *Active/Default*.

2.4.2.4 Сборка проекта

2.4.2.4.1 Введите в окна текстового редактора подготовленные исходные тексты: программы на языке С *main.c*, тестируемой функции *test.c* и библиотечной функции на языке ассемблера *lib.asm*.

2.4.2.4.2 В окно *C5515.cmd* скопируйте командный файл компоновщика из Приложения Е.

2.4.2.4.3 Выполните сборку проекта (см. пп. 1.4.2.2).

2.4.2.5 Профилирование программы

2.4.2.5.1 Непосредственно после запуска отладчика (см. пп. 1.4.2.4), задайте режим профилирования, выбрав пункт меню *Tools – Profile – Setup Profile Data Collection*. В появившемся окне *Profile Setup* нажмите кнопку *Activate*, установите флаг *Profile all Function for CPU Cycles* и сохраните произведенные изменения, нажав кнопку *Save*.

2.4.2.5.2 Откройте окно результатов профилирования *Profile*, выбрав пункт меню *Tools – Profile – Sview Function Profile Result*.

2.4.2.5.3 Запустите программу, выбрав пункт меню *Run – Resume*, и дождитесь остановки ее выполнения в режиме отладки.

2.4.2.5.4 В окне профилирование *Profile* проконтролируйте появление таблицы с результатами профилирования. Для сохранения результатов профилирования в формате электронной таблицы *.csv* выберите пункт контекстному меню *Data – Export All* и укажите имя и размещения файла с данными профилирования.

2.4.2.5.5 Завершите отладку, выбрав пункт меню *Run – Terminate*.

2.4.2.5.6 При анализе данных профилирования следует учесть назначение колонок:

– *Name* – задается имя профилируемой функции;

– *Calls* – число вызовов функции;

– *Excl Count Min (Max, Average, Total)* – максимальное (минимальное, среднее, общее) число циклов выполнения функции, исключая число циклов, необходимое для выполнения вызываемых в ее теле других функций;

– *Incl Count Min (Max, Average, Total)* – максимальное (минимальное, среднее, общее) число циклов выполнения функции, включая число циклов, необходимое для выполнения вызываемых в ее теле других функций;

– *Filename* – имя файла функции;

– *Line Number* – номер строки в файле;

– *Start Address* – начальный адрес программы в памяти.

2.4.3 Требования к отчету

Отчет по домашнему заданию должен содержать:

– титульный лист (приложение В);

– содержание;

– описание задания;

– результаты выполнения задания;

– выводы и рекомендации;

– список использованной литературы;

– приложения с текстами программ.

Оформление отчета осуществляется в соответствии с ГОСТ 2.105-95 [8], список литературы – по ГОСТ 7.1-2003 [9].

2.5 Контрольные вопросы 2

Контроль по модулю 2 заключается в объяснении директив подпрограммы обработки сигналов из стандартной библиотеки, обосновании разработанной программы реализации обработки сигналов на языке С, объяснении полученных экспериментальных данных и защиты сделанных выводов. Для проверки готовности к модульному контролю рекомендуется ответить на следующие контрольные вопросы.

Вопрос 2.1. Проверьте следующие три фрагменты программы на предмет корректного распараллеливания команд. Исправьте найденные ошибки.

```
a:  MOV *AR1+, AC1
    :: ADD @x, AR2
b:  MOV AC0, dbl(*AR2+)
    :: MOV dbl(*(AR1+T0)), AC2
c:  MPY *AR1+, *AR2+, AC0
    :: MPY *AR3+, *AR2+, AC1
```

Вопрос 2.2. Задан аналоговый сигнал $\chi(t) = 6\sin^2(\pi t/100)$ В. Какой необходимо выбрать шаг квантования во времени и шаг квантования по уровню, чтобы восстановить этот сигнал без искажений?

Вопрос 2.3. Задан аналоговый аудио-сигнал амплитудой 1 В и диапазоном частот от 50 Гц до 10 кГц и. Какая должна быть минимальная частота дискретизации и число уровней квантования, чтобы стало возможной восстановление исходного аналогового сигнала по его отсчетам? Что произойдет, если частота дискретизации будет 8 кГц? Какая сложится ситуация, если частота дискретизации будет 50 кГц?

Вопрос 2.4. Имеется одноканальное устройство цифровой обработки звука с частотой дискретизации 44,1 кГц и 2^{16} уровнями дискретизации. Какой объема памяти потребуется, чтобы записать аудио-сигнал длительностью одна минута?

Вопрос 2.5. Какие средства оптимизации программ на языке С использовались в лабораторной работе и почему?

Вопрос 2.6. Какие средства ускорения выполнения программы применены в библиотечной функции и какой эффект от их применения?

Вопрос 2.7. Обоснуйте методику разработки программы для цифровой обработки сигналов с использованием высокоуровневых и низкоуровневых средств программирования.

Список литературы

[1] Айфичер Э.С., Джервис Б.У. Цифровая обработка сигналов: практический подход / 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 992 с.

[2] Амосов В.В. Схемотехника и средства проектирования цифровых устройств. – СПб.: БХВ-Петербург, 2007. – 542 с.

[3] Безуглов Д.А., Калиенко И.В. Цифровые устройства и микропроцессоры – М.: Феникс, 2008. – 469 с.

[4] Бойко В.И. Схемотехника электронных схем. Микропроцессоры и микроконтроллеры. – СПб: БХВ-Петербург – Телеком, 2004. – 464 с.

[5] Болл С.Р. Аналоговые интерфейсы микроконтроллеров. – М.: Додэка-XXI, 2007. – 360 с.

[6] Быков Р.Е. Цифровое преобразование изображений. – М.: Горячая линия – Телеком, 2003. – 228 с.

[7] Витязев В.В., Витязев С.В. Цифровые процессоры обработки сигналов TMS320C67x компании Texas Instruments: Учебное пособие. – Рязань, 2007. – 112 с.

[8] ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам. – М.: Изд-во стандартов, 2012. – 26 с.

[9] ГОСТ 7.1–2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Изд-во стандартов, 2004. – 54 с.

[10] Керниган Б., Ритчи Д. Язык программирования Си / Пер. с англ., 3-е изд., испр. – СПб.: Невский диалект, 2003. – 352 с.

[11] Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительных систем. – СПб.: СПбГУ ИТМО, 2009. – 212 с.

[12] Корнеев В.В., Киселёв А.В. Современные микропроцессоры. – СПб.: БХВ-Петербург, 2003. – 440 с.

[13] Костров Б.В., Ручкин В.Н. Архитектура микропроцессорных систем. – М.: Диалог-МИФИ, 2007. – 304 с.

[14] Куприянов М.С., Матюшкин Б.Д. Цифровая обработка сигналов: процессоры, алгоритмы, средства проектирования. – 2-е изд., перераб. и доп. – СПб: Политехника, 1999. – 592 с.

[15] Лайсон Р. Цифровая обработка сигналов – М.: Бином-Пресс, 2006. – 656 с.

[16] Лэй Э. Цифровая обработка сигналов для инженеров и технических специалистов: практическое руководство – М.: Группа ИДТ, 2007. – 336 с.

[17] Методы компьютерной обработки изображений / Под ред. В.А. Сойфера – М.: ФИЗМАТЛИТ, 2003. – 784 с.

[18] Оппенгейм А., Шафер Р. Цифровая обработка сигналов. – М.: Техносфера, 2006. – 856 с.

[19] Проектирование систем цифровой и смешанной обработки сигналов / Под ред. У. Кестера; Пер. с англ. под ред. А.А. Власенко. – М.: Техносфера, 2010. – 328 с.

[20] Сергиенко А.Б. Цифровая обработка сигналов. – СПб.: Питер, 2002. – 606 с.

[21] Солонина А.И., Улахович Д.А., Арбузов С.М., Соловьева Е.Б. Основы цифровой обработки сигналов. – СПб: БХВ-Петербург, 2005. – 768 с.

[22] Сперанский В.С. Сигнальные процессоры и их применение в системах телекоммуникации и электроники. – М.: Горячая линия – Телеком, 2008. – 168 с.

[23] Страуструп Б. Язык программирования C++. Специальное издание / Пер. с англ. – М.: Издательство Бином, 2011. – 1054 с.

[24] Умняшкин С.В. Теоретические основы цифровой обработки и представления сигналов. – М.: ИНФРА-М, 2009. – 304 с.

[25] Arbona J. Design and Configuration Guide for the TLV320AIC3204 and TLV320AIC3254 Audio Codecs: Application Report. – Texas Instruments, 2010. – 29 p.

[26] Arbona J., Agarwal U. Audio Serial Interface Configurations for Audio Codecs. – Texas Instruments, 2010. – 6 p.

[27] Chassaing R. Digital Signal Processing. Laboratory Experiments Using C and the TMS320C31 DSK. – Chichester: Wiley, 1999. – 263 p.

[28] Chassaing R., Reay D. Digital Signal Processing with the TMS320C6713 and TMS320C6416 DSK. – Chichester: Wiley-Intersience, 2008. – 576 p.

[29] DAFX – Digital Audio Effects / Ed. Udo Zolzer. – Chichester: John Wiley & Sons, 2002. – 554 p.

[30] Kang N. Using the Cache Analysis Tool to Improve I-Cache Utilization on C55x Targets: Application Report. – Texas Instruments, 2003. – 20 p.

[31] Kehtarnavaz N. Real-Time Digital Signal Processing Based on the TMS320C6000. – Oxford: Elsevier, 2005. – 306 p.

[32] Kuo S.M., Lee B.H., Tian W. Real-time Digital Signal Processing. Implementations and Applications. – Chichester: Wiley, 2006. – 646 p.

[33] McKeown M. FFT Implementation on the TMS320VC5505, TMS320C5505 and TMS320C5515 DSPs: Application Report. – Texas Instruments, 2013. – 28 p.

[34] TMS320 DSP/BIOS 5.41: User's Guide. – Texas Instruments, 2009. – 306 p.

[35] TMS320C55x Technical Overview. – Texas Instruments, 2001. – 287 p.

[36] TMS320C55x Programmer's Guide. – Texas Instruments, 2001. – 287 p.

[37] TMS320C55x. User's Guide. – Texas Instruments, 1998. – 774 p.

[38] TMS320C55x CPU. Reference Guide. – Texas Instruments, 2009. – 265 p.

[39] TMS320C55x DSP: Functional Overview. – Texas Instruments, 2000. – 56 p.

[40] TMS320C55x DSP: Peripherals Overview: User's Guide. – Texas Instruments, 2011. – 27 p.

[41] TMS320C55x Hardware Extensions for Image/Video Applications: Programmer's Reference. – Texas Instruments, 2002. – 77 p.

[42] TMS320C55x Instruction Set Simulator: Technical Reference. – Texas Instruments, 2005. – 50 p.

[43] TMS320C55x Assembly Language Tools: User's Guide. – Texas Instruments, 2011. – 366 p.

[44] TMS320C55x Optimizing C/C++ Compiler: User's Guide. – Texas Instruments, 2011. – 181 p.

[45] TMS320C55x Mnemonic Instruction Set: Reference Guide. – Texas Instruments, 2009. – 863 p.

[46] TMS320C55x Algebraic Instruction Set: Reference Guide. – Texas Instruments, 2009. – 826 p.

[47] TMS320C55x DSP Library: Programmer's Reference. – Texas Instruments, 2009. – 144 p.

[48] TMS320C55x Image/Video Processing Library: Programmer's Reference. – Texas Instruments, 2004. – 95 p.

[49] TMS320C5515 Fixed-Point Digital Signal Processor. – Texas Instruments, 2011. – 160 p.

[50] TMS320C5515 DSP System: User's Guide. – Texas Instruments, 2011. – 82 p.

[51] TMS320C5515 Direct Memory Access Controller: User's Guide. – Texas Instruments, 2010. – 32 p.

[52] TMS320C5515 External Memory Interface: User's Guide. – Texas Instruments, 2011. – 92 p.

[53] TMS320C5515 Real Time Clock: User's Guide. – Texas Instruments, 2010. – 35 p.

[54] TMS320C5515 Timer/Watchdog Timer: User's Guide. – Texas Instruments, 2009. – 21 p.

[55] TMS320C5515 General-Purpose Input-Output: User's Guide. – Texas Instruments, 2009. – 20 p.

[56] TMS320C5515 Serial Peripheral Interface: User's Guide. – Texas Instruments, 2009. – 32 p.

[57] TMS320C5515 Universal Asynchronous Receiver/Transmitter: User's Guide. – Texas Instruments, 2009. – 37 p.

[58] TMS320C5515 Universal Serial Bus 2.0 Controller: User's Guide. – Texas Instruments, 2010. – 135 p.

[59] TMS320C5515 Successive Approximation Register. Analog-to-Digital Converter: User's Guide. – Texas Instruments, 2011. – 26 p.

[60] TMS320C5515 Inter-IC Sound Bus: User's Guide. – Texas Instruments, 2011. – 42 p.

[61] TMS320C5515 Inter-Integrated Circuit Peripheral: User's Guide. – Texas Instruments, 2009. – 39 p.

[62] TMS320C5515 Liquid Crystal Display Controller: User's Guide. – Texas Instruments, 2009. – 36 p.

[63] TMS320C5515 Multimedia Card/Secure Digital Card Controller: Reference Guide. – Texas Instruments, 2010. – 61 p.

[64] TMS320C5515 Evaluation Module: Technical Reference. – Spectrum Digital, 2010. – 76 p.

[65] TLV320AIC3204. Ultra Low Power Stereo Audio Codec. – Texas Instruments, 2008. – 159 p.

[66] DSP/BIOS Driver: Developer’s Guide. – Texas Instruments, 2005. – 123 p.

[67] Analysis Toolkit v1.3 for Code Composer Studio: User’s Guide. – Texas Instruments, 2005. – 52 p.

[68] Code Composer Studio Development Tools v3.3. Getting Started Guide. – Texas Instruments, 2006. – 103 p.

Приложение А

Отображаемые в память регистры

Адрес	Регистр	Разряды	Описание
000000h	IER0	15-0	Регистр разрешения прерываний 0
000001h	IFR0	15-0	Регистр флагов прерываний 0
000002h	ST0	15-0	Регистр статуса микропроцессора 0
000003h	ST1	15-0	Регистр статуса микропроцессора 1
000004h	ST3	15-0	Регистр статуса микропроцессора 3
000005h	–	–	Зарезервировано
000006h	–	–	Используется в режиме C54CM (ST0)
000007h	–	–	Используется в режиме C54CM (ST1)
000008h	AC0L	15-0	Младшее слово регистра-аккумулятора 0
000009h	AC0H	31-16	Старшее слово регистра-аккумулятора 0
00000Ah	AC0G	39-32	Сторожевой байт регистра-аккумулятора 0
00000Bh	AC1L	15-0	Младшее слово регистра-аккумулятора 1
00000Ch	AC1H	31-16	Старшее слово регистра-аккумулятора 1
00000Dh	AC1G	39-32	Сторожевой байт регистра-аккумулятора 1
00000Eh	T3	15-0	Временный регистр 3
00000Fh	TRN0	15-0	Переходный регистр 0
000010h	AR0	15-0	Дополнительный регистр 0
000011h	AR1	15-0	Дополнительный регистр 1
000012h	AR2	15-0	Дополнительный регистр 2
000013h	AR3	15-0	Дополнительный регистр 3
000014h	AR4	15-0	Дополнительный регистр 4
000015h	AR5	15-0	Дополнительный регистр 5
000016h	AR6	15-0	Дополнительный регистр 6
000017h	AR7	15-0	Дополнительный регистр 7
000018h	SP	15-0	Указатель стека данных
000019h	BK03	15-0	Регистр размера циклического буфера для AR0-AR3
00001Ah	BRC0	15-0	Счетчик блока повторения 0
00001Bh	RSA0L	15-0	Младшее слово начального адреса блока повторения 0
00001Ch	REA0L	15-0	Младшее слово конечного адреса блока повторения 0
00001Dh	–	–	Используется в режиме C54CM (ST3)
00001Eh	–	–	Зарезервировано
00001Fh	–	–	Зарезервировано
000020h	T0	15-0	Временный регистр 0
000021h	T1	15-0	Временный регистр 1
000022h	T2	15-0	Временный регистр 2
000023h	T3	15-0	Временный регистр 3
000024h	AC2L	15-0	Младшее слово регистра-аккумулятора 2

Адрес	Регистр	Разряды	Описание
000025h	AC2H	31-16	Старшее слово регистра-аккумулятора 2
000026h	AC2G	39-32	Сторожевой байт регистра-аккумулятора 2
000027h	CDP	15-0	Указатель коэффициентов
000028h	AC3L	15-0	Младшее слово регистра-аккумулятора 3
000029h	AC3H	31-16	Старшее слово регистра-аккумулятора 3
00002Ah	AC3G	39-32	Сторожевой байт регистра-аккумулятора 3
00002Bh	DPH	23-16	Старшая часть регистра страницы данных
00002Ch	–	–	Зарезервировано
00002Bh	–	–	Зарезервировано
00002Eh	DP	15-0	Регистр страницы данных
00002Fh	PDP	8-0	Регистр страницы в пространстве ввода-вывода
000030h	BK47	15-0	Регистр размера циклического буфера для AR4-AR7
000031h	BKC	15-0	Регистр размера циклического буфера для CDP
000032h	BSA01	15-0	Начальный адрес циклического буфера для AR0-AR1
000033h	BSA23	15-0	Начальный адрес циклического буфера для AR2-AR3
000034h	BSA45	15-0	Начальный адрес циклического буфера для AR4-AR5
000035h	BSA67	15-0	Начальный адрес циклического буфера для AR6-AR7
000036h	BSAC	15-0	Начальный адрес циклического буфера для CDP
000037h	–	–	Используется базовой системой ввода-вывода BIOS
000038h	TRN1	15-0	Переходный регистр 0
000039h	BRC1	15-0	Счетчик блока повторения 1
00003Ah	BRS1	15-0	Регистр для сохранения BRC1
00003Bh	CSR	15-0	Регистр простого повторения
00003Ch	RSA0H	23-16	Старший байт начального адреса блока повторения 0
00003Dh	RSA0L	15-0	Младшее слово начального адреса блока повторения 0
00003Eh	REA0H	23-16	Старший байт конечного адреса блока повторения 0
00003Fh	REA0L	15-0	Младшее слово конечного адреса блока повторения 0
000040h	RSA1H	23-16	Старший байт начального адреса блока повторения 1
000041h	RSA1L	15-0	Младшее слово начального адреса блока повторения 1
000042h	REA1H	23-16	Старший байт конечного адреса блока повторения 1
000043h	REA1L	15-0	Младшее слово конечного адреса блока повторения 1
000044h	RPTC	15-0	Регистр счетчика простого повторения
000045h	IER1	15-0	Регистр разрешения прерываний 1
000046h	IFR1	15-0	Регистр флагов прерываний 1
000047h	DBIER0	15-0	Регистр разрешения отладочных прерываний 0
000048h	DBIER1	10-0	Регистр разрешения отладочных прерываний 1
000049h	IVPD	15-0	Регистр адреса векторов прерываний 0-15, 24-31
00004Ah	IVPH	15-0	Регистр адреса векторов прерываний 15- 23
00004Bh	ST2	15-0	Регистр статуса микропроцессора 2
00004Ch	SSP	15-0	Регистр указателя системного стека

Адрес	Регистр	Разряды	Описание
00004Dh	SP	15-0	Регистр указателя стека данных
00004Eh	SPH	6-0	Регистр старшего байта указателей стеков
00004Fh	CDPH	6-0	Регистр старшего байта адреса коэффициентов
000050h	–	–	Зарезервировано
000051h	–	–	Зарезервировано
000052h	–	–	Зарезервировано
000053h	–	–	Зарезервировано
000054h	–	–	Зарезервировано
000055h	–	–	Зарезервировано
000056h	–	–	Зарезервировано
000057h	–	–	Зарезервировано
000058h	–	–	Зарезервировано
000059h	–	–	Зарезервировано
00005Ah	–	–	Зарезервировано
00005Bh	–	–	Зарезервировано
00005Ch	–	–	Зарезервировано
00005Dh	–	–	Зарезервировано
00005Eh	–	–	Зарезервировано
00005Fh	–	–	Зарезервировано

Приложение В

Титульный лист

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет ИУ – «Информатика и управление»

Кафедра ИУ-3 – «Информационные системы и телекоммуникации»

Отчёт по лабораторной работе 1
по дисциплине «Микропроцессорные устройства обработки сигналов»
на тему «Исследование поразрядных логических команд микропроцессора»

Студент группы ИУ3-72 (подпись) А.А. Иванов
12.09.2014

Преподаватель кафедры ИУ-3 (подпись) (И.О. Фамилия)
(дата)

Москва, 2014

Приложение С

Исходный текст модуля main.c

```
/* Включаемые файлы */
#include "stdio.h"

/* Макроопределения */
#define RESULTS 16

/* Глобальные данные */
long long d = 0xFEDCBE9876;
int r[RESULTS];

/* Объявление внешней функции */
extern long test(int*, int);

/* Точка входа в программу */
void main(void)
{
    /* Локальные переменные */
    int i, q;
    long t = 0x12345678;
    /* Подготовка исходных данных в формате Q15 */ )
#   pragma MUST_ITERATE(RESULTS, RESULTS, 1)
    for( i = 0, q = 32767; i < RESULTS; i++ )
    {
        r[i] = q, q >>= 1; /* 0,5, 0,25, 0,125, ... */
    }
    /* Вызов внешней функции */
    t = test(r, RESULTS);
    /* Вывод результа */
    printf( "Возвращаемое значение test - %08x\n", t );
    printf( "Глобальная переменная d - %010x\n", d );
#   pragma MUST_ITERATE(RESULTS, RESULTS, 1)
    for( i = 0; i < RESULTS; i++ )
    {
        printf( "Слово %d - %04x\n", i, r[i] );
    }
}
```

Приложение D

Исходный текст модуля test.asm

```
; Объявление перекрестных ссылок
    .def    _test
    .ref    _d
; Секция инициализованных данных
    .data
w   .word   0FFFFh, 0FFA9h, 8765h, 4320h
; Секция кода
    .text
; Аргументы функции test:
;   AR0     - указатель int* на массив
;   T0      - длина массива int
; Состояние стека:
;   *SP(#0)  - локальная переменная функции
;   *SP(#1)  - сохраненный регистр T2
;   *SP(#2)  - сохраненный регистр AR5
;   *SP(#3)  - адрес возврата из функции
;   *SP(#4)  - первый аргумент, не поместившийся в регистрах
; Определение локальной переменной
    .asg    *SP(#0), var
; Точка входа в функцию test
_test:
    ; Использование указателя стека при прямой адресации
    .cpl_on
    ; Сохранение в стеке модифицируемых регистров
    PSH     mmap(ST1_55)
    PSH     mmap(ST2_55)
    PSH     T2, AR5
    ; Выделение в стеке памяти для локальной переменной
    AADD    #-1, SP
    ; Задание режима работы микропроцессора
    BSET    M40     ; ST1 40-битные операции АЛУ
    BSET    SXMD    ; ST1 расширение знака при загрузке
    BCLR    ARMS    ; ST2 сигнальный режим адресации
    .arms_off
    ; Сохранить длину массива в локальной переменной
    MOV     T0, var
    ; Занести длину массива в счетчик простого повторения
    ADD     #-1, T0
    MOV     T0, CSR
    ; Скопировать указатель на массив в AR5
    MOV     AR0, AR5
    ; Инициализировать T1 числом 2^(-15)
    MOV     #1, T1
    ; Выполнить CSR+1 раз запись и модификацию данных
    RPT     CSR
    MOV     T1, *AR5+
    ||SFTL  T1, #1
    ; Загрузить в AC0 данные из области памяти w
    MOV     #w, AR0
    MOV     dbl(*AR0+), AC1
    MOV     dbl(*AR0+), AC0
    MOV     AC1, mmap(AC0G)
```

```
; Прибавить к AC0 2
ADD      #2, AC0
; Сохранить AC0 в глобальной переменной типа long long
MOV      #_d, AR1
SFTS     AC0, #-32, AC1
MOV      AC1, dbl(*AR1+)
MOV      AC0, dbl(*AR1+)
; Освобождение локальной памяти
AADD     #1, SP
; Восстановление регистров из стека
POP      T3, AR5
POP      mmap(ST2_55)
POP      mmap(ST1_55)
; Возврат из подпрограммы, AC0 – возвращаемое значение long
MOV      #0, AC0
MOV      T1, HI(AC0)
RET
```

Приложение Е

Командный файл компоновщика C5515.cmd

```
/* Параметры компоновки */
-c                    /* Компоновка по правиплам C */
-stack    0x2000      /* Размер первого стека  */
-sysstack 0x1000      /* Размер второго стека */
-heap     0x2000      /* Размер области динамической памяти */
-u _Reset             /* Загрузка вектора прерывания по сбросу */

/* Карта памяти */
MEMORY {

PAGE 0:  /* Встроенная память программ и данных */
 MMR    (RW):   origin = 0x000000, length = 0x0000c0 /* Регистры */
 DARAM  (RWIX): origin = 0x0000c0, length = 0x00ff40 /* ~64 кВ   */
 SARAM0 (RWIX): origin = 0x010000, length = 0x010000 /*  64 кВ   */
 SARAM1 (RWIX): origin = 0x020000, length = 0x020000 /* 128 кВ   */
 SARAM2 (RWIX): origin = 0x040000, length = 0x00FE00 /*  64 кВ   */
 VECS   (RWIX): origin = 0x04FE00, length = 0x000200 /* 512 В    */
 ROM    (RX):   origin = 0xfE0000, length = 0x01FF00 /* 128 кВ   */
 RESET  (RX):   origin = 0xFFFF00, length = 0x000100 /* 256 В    */

PAGE 1:  /* Внешняя память */
 CS0 (RW):   origin = 0x050000, length = 0x7B0000 /* ~8 MB SDRAM */
 CS2 (RW):   origin = 0x800000, length = 0x400000 /*  4 MB Async */
 CS3 (RW):   origin = 0xC00000, length = 0x200000 /*  2 MB Async */
 CS4 (RW):   origin = 0xE00000, length = 0x100000 /*  1 MB Async */
 CS5 (RW):   origin = 0xF00000, length = 0x0E0000 /* ~1 MB Async */

PAGE 2:  /* Пространство ввода-вывода */
 IOPORT (RWI) : origin = 0x000000, length = 0x020000
}
/* Размещение секций */
SECTIONS {
 .text    >> SARAM1|SARAM2|SARAM0 /* Секция кода */
 .stack    > DARAM                /* Первый стек */
 .sysstack > DARAM                /* Второй стек */
 .data    >> DARAM|SARAM0|SARAM1  /* Инициализированные данные */
 .bss     >> DARAM|SARAM0|SARAM1  /* Неинициализированные данные */
 .const   >> DARAM|SARAM0|SARAM1  /* Константы */
 .sysmem   > DARAM|SARAM0|SARAM1  /* Динамическая память */
 .switch   > SARAM2               /* Память оператора switch */
 .cinit    > SARAM2               /* Таблица инициализации данных*/
 .pinit    > SARAM2               /* Таблица функций эпилога */
 .cio      > SARAM2               /* Буферная память ввода-вывода*/
 .args     > SARAM2               /* Аргументы main() */
 vectors   > VECS                 /* Векторы прерываний */
 .ioport   > IOPORT PAGE 2        /* Порты ввода-вывода */
 .sdram    > CS0 PAGE 1           /* Внешняя синхронная память */
 .flash   >> CS2|CS3|CS4|CS5 PAGE 1 /* Внешняя асинхронная память */
}
```

Приложение F

Пример библиотечной функции

```
;*****************************************************************
; Версия XX.YY.ZZ
;*****************************************************************
; Функция gamma.
; Микропроцессор C55xx.
; Описание: криптографическое гаммирование данных.
; Алгоритм:
;   for(i = 0; i < nx; i++)
;      r[i] = x[i] ^ y[i];
; Использование:
;   gamma (
;        DATA *x,                  /* AR0 – адрес первого вектора */
;        DATA *y,                  /* AR1 – адрес второго вектора */
;        DATA *r,                  /* AR3 – адрес результата*/
;        ushort nx                 /* T0 – длина векторов */
;   );
;*****************************************************************
        .ARMS_off                       ;Сигнальный режим адресации
        .CPL_on                         ;Режим компилятора
        .mmregs                         ;Разрешение MMR-регистров
;-----------------------------------------------------------------
; Описание структуры стека данных
;-----------------------------------------------------------------
LOC_SZ  .set    1                       ;Объем локальных переменных
SAV_SZ  .set    1                       ;Объем сохраняемых регистров
RET_SZ  .set    1                       ;Размер адреса возврата
ARGS    .set    LOC_SZ+SAV_SZ+RET_SZ    ;Смещение аргументов
ARG_SZ  .set    0                       ;Объем аргументов
;-----------------------------------------------------------------
; Макроопределения для используемых регистров
;-----------------------------------------------------------------
        .asg    AR0, x_ptr              ;Первый вектор
        .asg    AR1, y_ptr              ;Второй вектор
        .asg    AR2, r_ptr              ;Результирующий вектор
        .asg    BRC0, outer_cnt         ;Счетчик наружнего цикла
;-----------------------------------------------------------------
; Тело функции
;-----------------------------------------------------------------
        .def    _gamma
        .text
_gamma:
; Сохранение регистров статуса в стеке
        PSH     mmap(ST2_55)
        PSH     mmap(ST1_55)
; Выделение в стеке локальной памяти
        ASUB    #LOC_SZ, SP
; Сохранение в локальной памяти регистров
        MOV     outer_cnt, @0
; Конфигурирование микропроцессора
        OR      #04020h, mmap(ST1_55) ;Установка CPL и C54CM
        AND     #07F00h, mmap(ST2_55) ;Сброс ARMS и ARxLC
; Установвить счетчик повторения наружного блока
```

```
        SUB     #1, T0                  ; T0 = nx-1
        MOV     T0, outer_cnt           ; BRC0 = nx-1
; Начало наружного блока команд
        RPTBLOCAL loop                  ;Команда повторения блока
        MOV     *x_ptr+, AC0            ;Чтение элемента x(i)
        XOR     *y_ptr+, AC0            ;Гаммирование с y(i)
loop:   MOV     LO(AC0), *r_ptr+        ;Запись результата
; Восстановить регистры из локальной памяти
        MOV     @0, outer_cnt
; Освободить локальную память
;       AADD    #LOC_SZ, SP
; Восстановление регистров статуса из стека
        POP     mmap(ST1_55)
        POP     mmap(ST2_55)
; Возврат из функции
        RET
;-----------------------------------------------------------------------
```

Приложение G

Программа умножения векторов комплексных чисел

```
#include "stdio.h"
#define len 3
/* Векторы A, B, C и D комплесных чисел длины 3*/
int VecA[2*len]={0x4000, 0x2000, 0x1000, 0x0800, 0x0400, 0x0200};
int VecB[2*len]={0x8000, 0xC000, 0xE000, 0xF000, 0xF800, 0xFC00};
int VecC[2*len], VecD[2*len];
/* Объявление внешних и внутренних функция */
extern void cmul(int*, int*, int*, int);
void cmultiply(int*, int*, int*, int);
/* Программа тестирования функций умножения векторов */
void main(void)
{
    /* Локальные переменные главной функции */
    short i, j;              /* Переменные цикла */
    short oflag;             /* Флаг переполения */
    /* Умножение векторов библиотечной функцией */
    oflag = cmul(VecA, VecB, VecC, len);
    /* Вывод образцовых результатов */
    printf("Overflow %d\n", oflag);
    for(i=0, j=0; i < len; i++, j+=2)
    {
        printf("%d - (%d, %d)\n", i, VecC[j], VecC[j+1]);
    }
    /* Умножение векторов исследуемой функцией */
    oflag = cmultiply(VecA, VecB, VecD, len);
    /* Вывод тестовых результатов */
    printf("Overflow %d\n", oflag);
    for(i=0, j=0; i < len; i++, j+=2)
    {
        printf("%d - (%d, %d)\n", i, VecD[j], VecD[j+1]);
    }
}
/* Тестируемая функция умножение векторов комплексных чисел*/
short cmultiply (int* VecA, int* VecB, int* VecC, int lenght )
{
    /* Локальные переменные */
    int i, j;
    int size = length << 1;
    short oflag = 0;
    /* Основной цикл */
#   pragma MUST_ITERATE (1)
    for(i = 0, j = 1; i < size; i += 2, j += 2)
    {
        /* Вычисление действительной части */
        VecC[i] = (VecA[i] * VecB[i])>>15
            - (VecA[j] * VecB[j])>>15;
        /* Вычисление мнимой части */
        VecC[j] = (VecA[i] * VecB[j])>>15
            + (VecA[j] * VecB[i])>>15;
    }
    return oflag;
}
```

Приложение Н

Функция умножения векторов комплексных чисел

```
;*****************************************************************
; Версия XX.YY.ZZ
;*****************************************************************
; Функция cmul.
; Микропроцессор C55xx.
; Описание: умножение векторов комплексных чисел в формате Q1.15.
; Алгоритм:
;    for(i = 0; i < nx; i++) {
;        Re r[j] = Re x[j] * Re y[j] - Im x[j] * Im y[j];
;        Im r[j] = Re x[j] * Im y[j] + Im x[j] * Re y[j];
;    }
; Использование:
;    short oflag =                  /* T0 – флаг перепонения */
;    cmul(
;        DATA *x,                   /* AR0 – адрес первого вектора */
;        DATA *y,                   /* AR1 – адрес второго вектора */
;        DATA *r,                   /* AR3 – адрес результата*/
;        ushort nx                  /* T0 – длина векторов */
;    );
;*****************************************************************
        .ARMS_off                       ; Сигнальный режим адресации
        .CPL_on                         ; Режим компилятора
        .mmregs                         ; Разрешение MMR-регистров
;-----------------------------------------------------------------
; Описание структуры стека данных
;-----------------------------------------------------------------
LOC_SZ  .set    1                       ; Объем локальных переменных
SAV_SZ  .set    4                       ; Объем сохраняемых регистров
RET_SZ  .set    1                       ; Размер адреса возврата
ARGS    .set    LOC_SZ+SAV_SZ+RET_SZ    ; Смещение аргументов функции
ARG_SZ  .set    0                       ; Объем аргументов функции
;-----------------------------------------------------------------
; Макроопределения для используемых регистров
;-----------------------------------------------------------------
        .asg    AR0, x_ptr              ; Первый вектор
        .asg    AR1, y_ptr              ; Второй вектор
        .asg    AR2, r_ptr              ; Результирующий вектор
        .asg    BRC0, outer_cnt         ; Счетчик наружнего цикла
        .asg    T0, oflag               ; Возвращаемое значение
;-----------------------------------------------------------------
; Тело подпрограммы
;-----------------------------------------------------------------
    .text
    .def    _cmul
_cmul:
; Сохранение регистров статуса в стеке
        PSH     mmap(ST3_55)
        PSH     mmap(ST2_55)
        PSH     mmap(ST1_55)
        PSHBOTH XCDP
; Выделение в стеке локальной памяти
        ASUB    #LOC_SZ, SP
```

```
; Сохранение в локальной памяти регистров
        MOV     outer_cnt, @0
; Конфигурирование микропроцессора
        AND     #001FFh, mmap(ST0_55) ; Сброс AC0Vx, TC1, TC2, CARRY
        OR      #04020h, mmap(ST1_55) ; Установка CPL, SXMD, FRCT
        AND     #0F9DFh, mmap(ST1_55) ; Сброс M40, SATD, C54CM
        AND     #07A00h, mmap(ST2_55) ; Сброс ARMS,RDM, CDPLC,ARxLC
        AND     #0FFDDh, mmap(ST3_55) ; Сброс SATA, SMUL
; Установвить счетчик повторения наружного блока
        SUB     #1, T0                ; T0 = nx-1
        || BCC  L2, T0 < #0           ; Переход по нулевой длине
        MOV     T0, outer_cnt         ; BRC0 = nx-1
        || MOV  #1, T0                ; В T0 смещение мнимой части
        AMOV    XAR1, XCDP            ; Указатель на вектор y в XCDP
        ||MOV   #2, T1                ; В T1 размер числа
; Начало наружного блока команд
        RPTBLOCAL L0                  ; Команда повторения блока
        MPY     *AR0, *CDP+, AC0      ; AC0 = Re x[i]*Re y[i]
        :: MPY  *AR0(T0), *CDP+, AC1  ; :: AC1 = Im x[i]*Re y[i]
        MAS     *AR0(T0), *CDP+, AC0  ; AC0 -= Im x[i]*Im y[i]
        :: MAC  *(AR0+T1), *CDP+, AC1 ; AC0 += Re x[i]*Im y[i]
L0:     MOV     pair(HI(AC0)), dbl(*AR1+) ; Запись x[i]*y[i] в r[i]
; Проверка на переполнение
        MOV     #0, oflag             ; Сброс флага переполнения
        XCC     L1, overflow(AC0)     ; Ветвление по ACOV0
        ||MOV   #1, oflag             ; Установка флага переполения
L1:     XCC     L2, overflow(AC1)     ; Ветвление по ACOV1
        ||MOV   #1, oflag             ; Установка флага переполения
; Восстановить регистры из локальной памяти
L2:     MOV     @0, outer_cnt
; Освободить локальную память
;       AADD    #LOC_SZ, SP
; Восстановление регистров из стека
        POPBOTH XCDP
        POP     mmap(ST1_55)
        POP     mmap(ST2_55)
        POP     mmap(ST3_55)
; Возврат из функции
        RET
;-----------------------------------------------------------------------
```

Приложение I

Оценочная плата TMS320C5515 Evaluation Module

Внешний вид оценочной платы приведен на рисунке ПI.1.

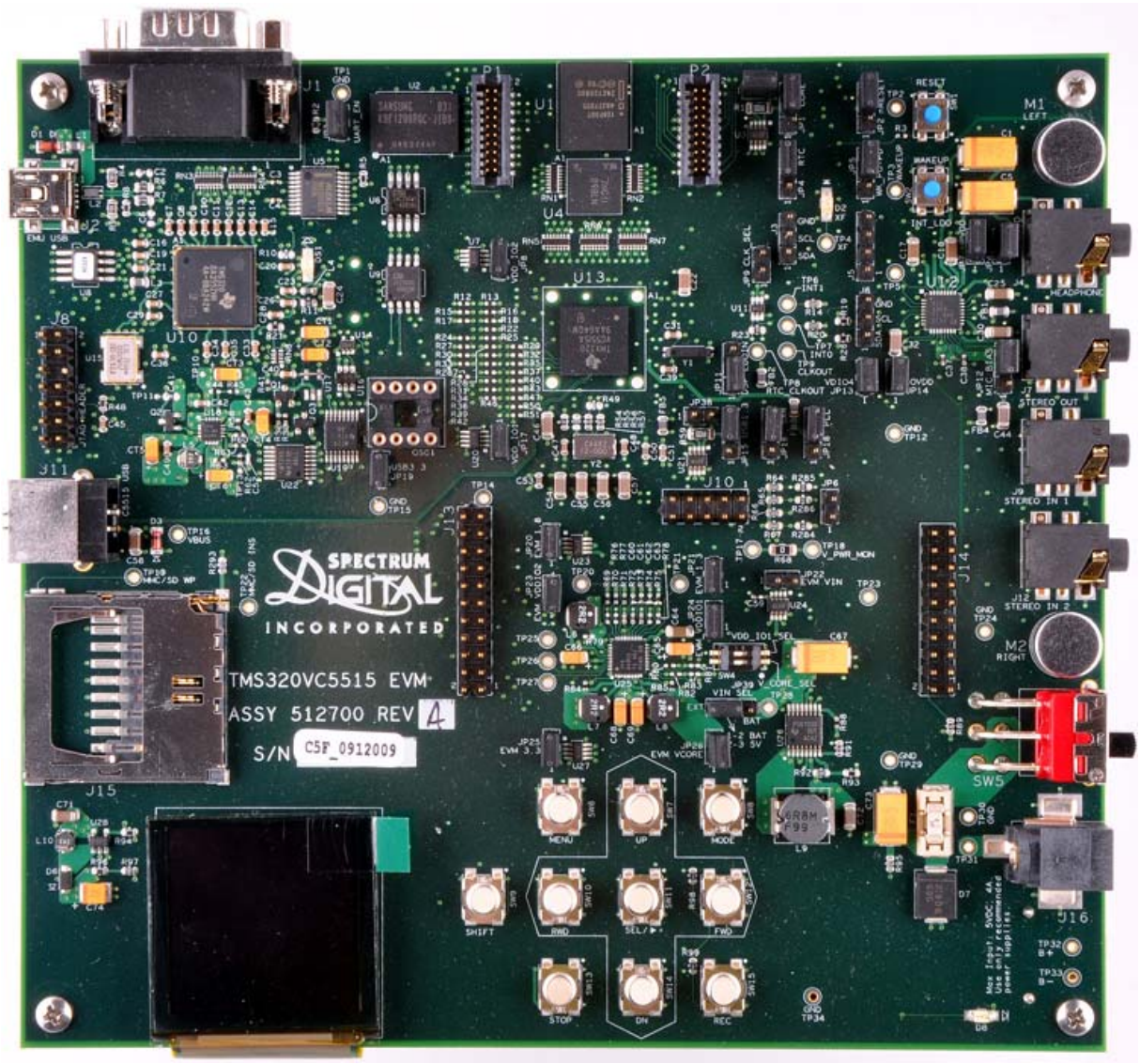


Рисунок ПI.1 – Оценочная плата TMS320C5515™ Evaluation Module

Для подключения платы к питающей сети и персональному компьютеру используются принадлежности, показанные на рисунке ПІ.2.

а) б)

Рисунок ПІ.2 – Блок питания и интерфейсный кабель USB

Микросхемы, смонтированные на оценочной плате, перечислены в таблице ПІ.1.

Таблица ПІ.1 – Микросхемы оценочной платы

Поз.	Наименование
U1	Флэш-память PC28F128P30T85 типа NOR, 128 МБит
U2	Флэш-память K9F1208R0B-JIB0 типа NAND, 64 МБит x 8
U3	Монитор тока и напряжения INA219AIDCN
U4	Мобильная синхронная динамическая память MT48H4M16LFB4-8, 64 МБит
U5	Передатчик интерфейса RS-232 MAX3222ECAP+
U6	Электрически перепрограммируемое ПЗУ CAT24C256WI-GT3, 256 кБит
U7	Монитор тока и напряжения INA219AIDCN
U9	Электрически перепрограммируемое ПЗУ CAT25256XI-T2, 256 кБит
U11	1-разрядный двунаправленный шинный формирователь SN74AVC1T45DCKR
U12	Стерео-кодек TLV320AIC3204IRHBT
U13	Цифровой сигнальный микропроцессор TMX320VC5515AZCH
U19	4-разрядный двунаправленный шинный формирователь SN74AVC4T245PWR
U20	Монитор тока и напряжения INA219AIDCN
U21	Монитор тока и напряжения INA219AIDCN
U22	4-разрядный двунаправленный шинный формирователь SN74AVC4T245PWR
U23	Монитор тока и напряжения INA219AIDCN
U24	Монитор тока и напряжения INA219AIDCN
U25	Микросхема TPS65023RSBR для управления аккумуляторами типа LI-ION

Таблица ПИ.1 – Микросхемы оценочной платы

Поз.	Наименование
U26	Преобразователь напряжения питания TPS61030PWP
U27	Монитор тока и напряжения INA219AIDCN
U28	Преобразователь напряжения питания TPS61041DBVR
U29	1-разрядный двунаправленный шинный формирователь SN74AVC1T45DCKR

ПИ.1 Карта памяти

Рисунок ПИ.3 – Карта памяти оценочной платы

ПИ.2 Исходное состояние перемычек

JP1 (CPU VDDC Select) в позиции 1–2;

JP2 (nRESET Select) в позиции 1–2;

JP3 (UART_EN) закорочен;

JP4 (RTC) в позиции 1–2;

JP5 (WK_PU_PD_SEL) в позиции 2–3;

JP6 (LDO_EN) закорочен;

JP9 (CLK_SEL) разорван;

JP11 (LDOI Select) в позиции 1–2;

JP12 (MIC_BIAS) в позиции 2–3;

JP15 (USB_VDD_IN Select) в позиции 1–2;

JP16 (VDDA_ANA Select) в позиции 1–2;

JP18 (VDDA_PLL Select) в позиции 1–2;

JP39 (VIN Select) в позиции 1–2.

Глоссарий

ADC – Analog-to-Digital Converter, аналого-цифровой преобразователь.

App-Spec – Application Specific, специфичный для приложений.

Bridge – мост, устройство согласования двух интерфейсов.

Bus – шина, магистраль.

Ceiling – округление к большему числу с заданной точностью, в частном случае – к большему целому.

Clock – тактовая частота, тактовое питание.

Connectivity – подключение, соединение.

Core – ядро, центральная часть микропроцессора.

CPU – Central Processing Unit, центральное процессорное устройство, центральный процессор.

DAC – Digital-to-Analog Converter, цифро-аналоговый преобразователь.

DARAM – Dual Access Random Access Memory, двухвходовая память со случайным доступом.

Display – устройство для отображения данных.

DMA – Direct Memory Access, прямой доступ к памяти.

DSP – Digital Signal Processor, цифровой сигнальный процессор.

EEPROM – Electrically Erasable Programmable Read Only Memory, электрически стираемое перепрограммируемое постоянное запоминающее устройство, флэш-память.

EHPI – Enhanced Host-Port Interface, усовершенствованный интерфейс для связи с другим процессором.

EMIF – External Memory Interface, интерфейс внешний памяти.

IFFT – Inverse Fast Fourier Transform, обратное быстрое преобразование Фурье.

Fix – округление к меньшему по модулю числу с заданной точностью, в частном случае – к целому числу;

FFT – Fast Fourier Transform, быстрое преобразование Фурье.

Floor – округление к меньшему числу с заданной точностью, в частном случае, к наименьшему целому.

GP– General Purpose, устройство общего назначения.

GPIO – General Purpose Input-Output, входы-выходы общего назначения.

HWAFFT – Hardware Accelerator of Fast Fourier Transform, аппаратурный ускоритель быстрого преобразования Фурье.

I2C – Inter-Integrated Circuit, межмикросхемный интерфейс.

I2S – Integrated Inter-chip Sound, интегрированный межмикросхемный интерфейс для передачи звука.

INT – Interrupt, прерывание.

Integer – округление к большему по модулю числу с заданной точностью, в частном случае – к целому числу.

Interconnect – внутреннее соединение, межкомпонентное соединение.

JTAG – Joint Test Action Group, специализированный аппаратурный интерфейс для тестирования и отладки сложных дискретных устройств.

LCD – Liquid crystal display, жидкокристаллический дисплей.

LDO – Low-Drop Out, досл. «низкое падение», источник стабилизированного питания с низким падением напряжения.

LSB – Least Significant Bit, менее значимый бит, бит с наименьшим весом.

Master – главное (ведущее) устройство на интерфейсе.

McBSP – Multichannel Buffered Serial Ports, многоканальный буферизированный последовательный порт.

MMC/SD – Multi Media Card or Secure Digital, мультимедийная карта или защищенная цифровая карта памяти.

MMR – Memory Mapped Register, регистр, доступный для чтения-записи через адресное пространство памяти.

MSB – Most Significant Bit, самый значимый бит, бит с наибольшим весом.

mSDRAM – mobile Synchronous Dynamic Random Access Memory, мобильная синхронная динамическая память со случайным доступом.

NAND Flash – Not-AND Flash, флэш-память на элементах И-НЕ.

NOR Flash – Not-OR Flash, флэш-память на элементах ИЛИ-НЕ.

PBGA – Plastic Ball Grid Array, пластиковый корпус интегральных микросхем для поверхностного монтажа с контактами в виде шариков припоя.

Peripherals – периферийные устройства.

Peripheral Bus – периферийная шина.

PLL – Phase-Locked Loop, фазовая автоподстройка частоты.

RAM – Random Access Memory, память со случайным доступом,

ROM – Read Only Memory, память только для чтения, постоянное запоминающее устройство.

Rounding – округление к ближайшему числу с заданной точностью (в частном случае – к целому числу).

RTC – Real-Time Clock, часы реального времени.

SAR – Successive Approximation Register, регистр последовательного приближения.

SARAM – Single Access Random Access Memory, одновходовая основная память.

Serial Interface – последовательный интерфейс.

SCR – Switched Central Resource, центральное коммутирующее устройство.

Slave – подчиненное, ведомое устройство на интерфейсе.

SPI – Serial Port Interface, порт последовательного интерфейса.

SRAM – Static Random Access Memory, статическая память со случайным доступом.

Storage – запоминающее устройство.

Timer – таймер, устройство для измерения и порождения интервалов времени.

UART – Universal Asynchronous Receiver-Transmitter, универсальный асинхронный приёмопередатчик.

USB – Universal Serial Bus, универсальная последовательная шина.

WD – Watch Dog (Timer), сторожевой таймер.

Электронное учебное издание

Учебное пособие

Выхованец Валерий Святославович

Демин Никита Александрович

Мозговая Екатерина Игоревна

Назарова Светлана Игоревна

Рожкова Диана Александровна

Шапкина Евгения Сергеевна

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА ОБРАБОТКИ СИГНАЛОВ