Л.Я. Яковлев

ЦИФРОВЫЕ ПРОЦЕССОРЫ ОБРАБОТКИ СИГНАЛОВ

Методические рекомендации к лабораторным работам 201100, 201200, 201400

Содержание:

ЛАБОРАТОРНАЯ РАБОТА 1

ПОДГОТОВКА ИСПОЛНЯЕМЫХ ПРОГРАММ ДЛЯ ЦПОС TMS320C50

1.1. Цель работы и порядок выполнения

Целью работы является получение практических навыков в подготовке исполняемых программ для ЦПОС TMS320C50.

Работа выполняется в следующем порядке:

- изучение п. 1.2;
- подготовка задания в соответствии с указаниями п. 1.3;
- выполнение работы в соответствии с указаниями п. 1.4.

1.2. Краткая теоретическая справка

Процесс подготовки исполняемых программ для ЦПОС TMS320C50 можно разделить на четыре следующих этапа:

1. Создание одного или нескольких модулей-файлов ***(имя файла).asm,*** с исходным текстом программы, написанным на языке ассемблера.
2. Трансляция созданного файла ***(имя файла).asm*** программой-транслятором ***DSPA.EXE*** и получение объектного файла ***(имя файла).obj*** и листинга ***(имя файла).lst.*** При трансляции помимо перевода текста программы производится проверка исходного текста программы на наличие синтаксических ошибок ассемблера, сообщения о которых приводятся в листинге.
3. Написание командного файла компоновки ***(имя файла).cmd*** с указанием компонуемых программных модулей, порядка соединения секций и используемых областей памяти для размещения этих секций.
4. Компоновка модулей и секций с помощью компоновщика ***DSPALNK.EXE*** и получение выходного файла ***(имя файла).out*** и файла ***(имя файла).map,*** позволяющего контролировать правильность распределения памяти и соединения секций, заданных в командном файле ***(имя файла).cmd.***

Примером программы, на базе которой будут отрабатываться основные шаги по подготовке исполняемых программ для ЦПОС TMS320C50, является программа ***SIM.ASM,*** используемая также в лабораторной работе 2 “Изучение симулятора ЦПОС TMS320C50 и основных приемов работы с ним”. Программа ***SIM*** состоит из нескольких частей, в каждой из которых реализуется некоторая операция. В программе ***SIM*** отдельные операции разделены метками, такими же, как и метки описаний операций, представленных ниже. В табл. 1.1 представлена карта размещения переменных, используемых в программе, в памяти данных процессора.

Таблица 1.1

Карта размещения переменных, используемых в программе ***SIM***, в памяти данных

Переменная	Адрес ячейки памяти данных (Hex)	Начальное значение переменной (Hex)
X	0800	1
Y	0801	3
Z	0802	0
X1	0803	4
Y1	0804	5
Z1	0805	0
X2	0806	0a
Y2	0807	0b
Z2	0808	0c

Программа также резервирует 6 ячеек памяти для записи промежуточных результатов. Первая из этих ячеек имеет символический адрес **M1.**

Ниже представлены операции, реализуемые в программе ***SIM***, и соответствующие им метки в тексте программы:

1. Операция сложения переменной **X** с переменной **Y** с сохранением результата в ячейке памяти **Z** (имена переменных воспринимаются транслятором как адреса ячеек памяти, где хранятся эти переменные).

L1:

```
LDP   #X      ; Указать на страницу памяти, хранящую X
LACC  X       ; Загрузить в ACC содержимое ячейки X
ADD   Y       ; Прибавить к ACC содержимое ячейки Y
ADD   #5      ; Прибавить к ACC непосредственное слагаемое
SACL  Z       ; Сохранить содержимое младшего слова ACC в ячейке Z
```

Здесь и далее фразу “Прибавить к ACC” в комментариях к командам следует понимать как “Прибавить к содержимому ACC”, подобная запись используется для сокращения количества слов.

2. Операция перемножения переменной **X1** и переменной **Y1** с сохранением результата в ячейке памяти **Z1**.

L2:

```
LDP    #X1     ; Указать на страницу памяти, хранящую X1
LT     X1      ; Загрузить в Т-регистр содержимое ячейки X1
MPY    Y1      ; Перемножить X1 и Y1. Результат в PREG
SPL    Z1      ; Сохранить содержимое младшего слова PREG в ячейке
               Z1
```

В приведенном примере учебной программы при выполнении отдельных операций каждый раз устанавливается требуемая страница памяти данных командой **LDP**. В реальных программах страница памяти устанавливается только тогда, когда она меняется. Если же при очередной операции она остается прежней (как при переходе к операциям под меткой **L2**), то нет необходимости ее указывать заново.

В приведенном фрагменте (метка **L2**) в качестве результата перемножения используется младшее слово содержимого регистра *PREG*. Оно сохраняется в памяти командой **SPL**. На самом деле в зависимости от того, какие числа (целые или дробные) используются, результат может быть как в младшем, так и старшем слове содержимого регистра *PREG*. Изучению представления данных в процессоре посвящена лабораторная работа 3. В данном случае считается, что используются целые положительные числа. Это замечание относится и к операции накопления результатов перемножения, приведенной под меткой **L8**.

3. Операция вычитания переменной **X1** из переменной **Y1** с сохранением результата в ячейке памяти **Z2**.

L3:

```
LDP    #X1     ; Указать на страницу памяти, хранящую X1
LACC   Y1      ; Загрузить в ACC содержимое ячейки Y1
SUB    X1      ; Вычесть из ACC содержимое ячейки X1
SACL   Z2      ; Сохранить содержимое младшего слова ACC в ячейке Z2
```

4. Операция присвоения переменной **X** значения **Z1**.

L4:

```
LDP     #X      ; Указать на страницу памяти, хранящую X
LACC    Z1      ; Загрузить в ACC содержимое ячейки Z1
SACL    X       ; Сохранить содержимое младшего слова ACC в ячейке X
```

5. Операции активизации вспомогательного регистра **AR0** и загрузки в него содержимого ячейки **Y2**.

```
L5:
LDP     #Y2     ; Указать на страницу памяти, хранящую X
MAR     *,AR0   ; Сделать активным регистр AR0
LAR     AR0,Y2  ; Загрузить в AR0 содержимое ячейки Y2
```

6. Операция загрузки во вспомогательный регистр **AR6** адреса ячейки **Z**.

```
L6:
LDP     #Z      ; Указать на страницу памяти, хранящую X
LAR     AR6,#Z  ; Загрузить в AR6 адрес ячейки Z
```

7. Операции загрузки числа **7FFF** (Hex) в аккумулятор и сохранения его в ячейке **Y2**, в буфере аккумулятора и во вспомогательном регистре **AR1**.

```
L7:
LDP     #Y2     ; Указать на страницу памяти, хранящую Y2
LACC    #07FFFh ; Загрузить в ACC число 7FFF (Hex)
SACL    Y2      ; Сохранить содержимое младшего слова ACC в ячейке Y2
SACB            ; Сохранить содержимое ACC в ACCB
SAMM    AR1     ; Сохранить содержимое ACC в регистре AR1
B       L9      ; Безусловный переход к фрагменту программы с меткой
                  L9
```

Переход к другому фрагменту программы демонстрирует действие команд переходов. В данном примере команды для выполнения всех необходимых действий можно записать последовательно.

8. Операции накопления результатов умножения и сохранения полученной суммы произведений.

L8:

```
LDP    #X      ; Указать на страницу памяти, хранящую X
ZAP            ; Обнулить ACC и регистр PREG
LT     X       ; Загрузить в регистр T переменную X
MPY    X1      ; Умножить X на X1, результат в регистре PREG
LTA    Y       ; Загрузить в регистр T переменную Y, предыдущее
               ; произведение из PREG добавляется в ACC
MPY    Y1      ; Умножить Y на Y1, результат в регистре PREG
LTA    Z       ; Загрузить в регистр T переменную Z, предыдущее
               ; произведение из PREG добавляется в ACC
MPY    Z1      ;Умножить Z на Z1, результат в регистре PREG
APAC           ; предыдущее произведение из PREG добавляется в ACC
SACL   X2      ; Сохранить содержимое младшего слова ACC в ячейке X2
```

9. Операция занесения констант в зарезервированные ячейки памяти.

L9:

```
LDP    #M1     ; Указать на страницу памяти с зарезервированной
               ; ячейкой M1
SPLK   #1,M1   ; Загрузить в M1 число 1
SPLK   #2,M1+1 ; Загрузить в (M1+1) число 2
SPLK   #3,M1+2 ; Загрузить в (M1+2) число 3
```

```
        SPLK    #4,M1+3   ; Загрузить в (M1+3) число 4

        SPLK    #5,M1+4   ; Загрузить в (M1+4) число 5

        SPLK    #6,M1+5   ; Загрузить в (M1+5) число 6

        B       L8        ; Безусловный переход к фрагменту программы с меткой
                          L8
```

Примечание. *Еще раз отметим, что так как все переменные, используемые в программе* ***SIM,*** *расположены на одной странице памяти, при обращении к ним в принципе достаточно в начале программы поставить один раз указатель страницы памяти, указывая на любую из этих переменных. Например:* **LDP X.**

Полный текст программы ***SIM,*** которая должна быть набрана и оттранслирована в ходе выполнения лабораторной работы, приводится ниже.

```
; Программа выполнения простейших операций на ЦПОС TMS320C50

        .version    50

        .mmregs             ; Разрешение использования символьных

                            ; имен регистров процессора.

; Задание значений переменных в памяти данных

        .data

X       .word       1       ; Начальное значение X
Y       .word       2       ; Начальное значение Y
Z       .word       0       ; Начальное значение Z
X1      .word       4       ; Начальное значение X1
Y1      .word       5       ; Начальное значение Y1
Z1      .word       0       ; Начальное значение Z1
X2      .word       0ah     ; Начальное значение X2
Y2      .word       0bh     ; Начальное значение Y2
Z2      .word       0ch     ; Начальное значение Z2

; Резервирование области памяти размером 6 ячеек

        .bss        M1,6    ; Символический адрес первой ячейки M1

; Таблица векторов прерывания
```

```
        .sect " Vectors "

RESET   B       START   ; Начальная строка таблицы векторов
                        ; прерывания – переход к основной
                        ; программе

; Основная программа

.text

START   LDP     #0      ; Указатель страницы памяти данных
        OPL     #04h,PMST ; Задать значение регистра PMST
        CLRC    OVM     ; Запретить режим переполнения
        SPM     0       ; Установить нулевой сдвиг PREG
L1:
        LDP     #X      ; Указать на страницу памяти, хранящую X
        LACC    X       ; Загрузить в ACC содержимое ячейки X
        ADD     Y       ; Прибавить к ACC содержимое ячейки Y
        ADD     #5      ; Прибавить к ACC непосредственное слагаемое
        SACL    Z       ; Сохранить содержимое младшего
                        ; слова ACC в ячейке Z
L2:
        LDP     #X1     ; Указать на страницу памяти, хранящую X1
        LT      X1      ; Загрузить в T-регистр содержимое ячейки X1
        MPY     Y1      ; Перемножить X1 и Y1, результат в PREG
        SPL     Z1      ; Сохранить содержимое младшего
                        ; слова PREG в ячейке Z1
L3:
```

```
        LDP     #X1     ; Указать на страницу памяти, хранящую X2
        LACC    Y1      ; Загрузить в ACC содержимое ячейки X2
        SUB     X1      ; Вычесть из ACC содержимое ячейки Y2
        SACL    Z2      ; Сохранить содержимое младшего слова ACC
                        ; в ячейке Z2
L4:
        LDP     #X      ; Указать на страницу памяти, хранящую X
        LACC    Z1      ; Загрузить в ACC содержимое ячейки Z1
        SACL    X       ; Сохранить содержимое младшего слова ACC
                        ; в ячейке X
L5:
        LDP     #Y2     ; Указать на страницу памяти, хранящую Y
        MAR     *,AR0   ; Сделать активным регистр AR0
        LAR     AR0,Y2  ; Загрузить в AR0 содержимое ячейки Y2
L6:
        LDP     #Z      ; Указать на страницу памяти, хранящую X
        LAR     AR6,#Z  ; Загрузить в AR6 адрес ячейки Z
L7:
        LDP     #Y2     ; Указать на страницу памяти, хранящую Y2
        LACC    #07fffh ; Загрузить в ACC число 7fff (Hex)
        SACL    Y2      ; Сохранить содержимое младшего слова ACC
                        ; в ячейке Y2
        SACB            ; Сохранить содержимое ACC в ACCB
        SAMM    AR1     ; Сохранить содержимое ACCв регистре AR1
```

```
        B       L9      ; Безусловный переход к фрагменту
                        ; программы с меткой L9
L8:
        LDP     #X      ; Указать на страницу памяти, хранящую X
        ZAP             ; Обнулить ACC и регистр PREG
        LT      X       ; Загрузить в регистр T переменную X
        MPY     X1      ; Умножить X на X1 результат в регистре PREG
        LTA     Y       ; Загрузить в регистр T переменную Y,
                        ; предыдущее произведение из PREG
                        ; добавляется в ACC
        MPY     Y1      ; Умножить Y на Y1 результат в регистре PREG
        LTA     Z       ; Загрузить в регистр T переменную Z,
                        ; предыдущее произведение из PREG
                        ; добавляется в ACC
        MPY     Z1      ; Умножить Z на Z1 результат в регистре PREG
        APAC            ; предыдущее произведение из PREG
                        ; добавляется в ACC
        SACL    X2      ; Сохранить содержимое младшего
                        ; слова ACC в ячейке X2
        B       START   ; Переход к началу программы для повторения
L9:
        LDP     #M1     ; Указать на страницу памяти
                        ; с зарезервированной ячейкой M1
        SPLK    #1,M1   ; Загрузить в M1 число 1
        SPLK    #2,M1+1 ; Загрузить в (M1+1) число 2
        SPLK    #3,M1+2 ; Загрузить в (M1+2) число 3
```

```
SPLK    #4,M1+3     ; Загрузить в (M1+3) число 4
SPLK    #5,M1+4     ; Загрузить в (M1+4) число 5
SPLK    #6,M1+5     ; Загрузить в (M1+5) число 6
B       L8          ; Безусловный переход к фрагменту программы
                    ; с меткой L8
.end                ; Конец программы
```

Для компоновки представленной выше программы необходимо написать командный файл ***SIM.CMD***, в котором задается распределение памяти ЦПОС TMS320C50. Одно из возможных распределений памяти, используемое при изучении процессора с помощью программного имитатора (симулятора) приведено в командном файле, представленном ниже:

```
–v0 /* версия (цифра 0) */
–e RESET
–m sim.map /* Задание имени файла с таблицей компоновки *. map */
–o sim.out /* Задание имени выходного файла *. out (буква o) */
sim.obj /* Компонуемый объектный файл,
сформированный программой dspa.exe */
MEMORY
{
PAGE 0: /* Память программ */
/* Блок для векторов прерываний в памяти */
VECS: origin = 0x0000, length = 0x02
/* Блок для размещения программ в памяти процессора */
PROG: origin = 0x0040, length = 0x0200
PAGE 1: /* Память данных */
/* Размещение регистров процессора, отраженных на память в памяти
```

процессора */

REGS: origin = 0x0000, length = 0x0060

/* Блок для размещения данных в памяти данных процессора */

BLOCK0: origin = 0x0800, length = 0x0200

}

SECTIONS

{

Vectors : { } > VECS PAGE 0

.text : { } > PROG PAGE 0

.data : { } > BLOCK0 PAGE 1

.bss : { } > BLOCK0 PAGE 1

/* Секции **.data** и **.bss** будут размещаться в памяти данных, начиная с адреса 800h подряд непосредственно друг за другом, занимая количество ячеек в соответствии со своими реальными размерами */

}

Таким образом, на основе объектного файла ***SIM.OBJ,*** полученного после трансляции, под управлением ***SIM.CMD*** программой-компоновщиком ***DSPLNK.EXE*** формируется выходной файл ***SIM.OUT***, который будет использоваться при работе с программой симулятора. Программа-компоновщик формирует также таблицу реально полученного распределения памяти ***SIM.MAP.*** Этот файл позволяет контролировать инициализацию начальных адресов и количество ячеек памяти, используемых для хранения программы и данных в процессоре, а также правильность расположения и соединения секций программы.

Примерный вид файла ***SIM.MAP:***

TMS 320C1x/2x/C2xx/C5x COFF Linker Version 6.60

OUTPUT FILE NAME : < sim.out >

/ Имя выходного файла

ENTRY POINT SYMBOL : “ RESET “ address : 0000 0000

/ Присвоение слову RESET конкретного адреса 0000 0000

MEMORY CONFIGURATION

/ Конфигурация памяти процессора в соответствии с заданием

/ в командном файле компоновки

name	origin	length	attributes	fill

PAGE 0: VECS 00000000 000000002 RWIX

PROG 00000040 000000200 RWIX

PAGE 1: REGS 00000000 000000060 RWIX

BLOCK0 00000800 000000200 RWIX

SECTION ALLOCATION MAP

/ Реально сформированные адреса размещения секций программного

/ файла в памяти процессора и количество ячеек памяти, занятых

/ в памяти процессора под ту или иную секцию

output section	page	origin	length	attributes/ input sections
Vectors	0	00000000	00000002	
		00000000	00000002	sim.obj (Vectors)
.text	0	00000040	00000032	
		00000040	00000032	
.data	1	00000800	00000009	
		00000800	00000009	sim.obj (.data)
.bss	1	00000000	00000000	UNINITIALIZED
		00000000	00000000	sim.obj (.bss)

GLOBAL SYMBOLS

00000000	.bss	00000000	end
00000800	.data	00000000	.bss
00000040	.text	00000000	RESET
00000000	RESET	00000040	START
00000040	START	00000040	.text
00000809	edata	00000072	etext
00000000	end	00000800	.data
00000072	etext	00000809	edata

[8 symbols]

1.3. Задание на самостоятельную подготовку

Изучите ЦПОС TMS320C50, его регистры состояния и управления и карту распределения памяти. При подготовке использовать материалы лекций и приложений A–D.

По материалам лекций и настоящему описанию, изучите простейшие команды процессора и программу ***SIM*** (см. п. 1.2)**.**

1.4. Выполнение лабораторной работы

1. В своей рабочей директории создать файл ***(имя файла).asm*** и набрать в нем текст программы ***SIM***, представленный в п. 1.2 “Краткая теоретическая справка”.
2. Скопировать в свой директорий файлы транслятора и компоновщика ***dspa.exe*** и ***dsplnk.exe.***
3. Оттранслировать полученный файл ***(имя файла).asm*** и получить объектный файл ***(имя файла).obj*** и листинг – ***(имя файла).lst*** с помощью программы ***dspa.exe*** . Для этого набрать в командной строке следующую команду:

   dspa.exe –lcs (имя файла).asm

   Если программа ***SIM*** набрана верно и ошибок в написании программы и при определении переменных не обнаружено, на экране монитора должна появиться следующая надпись:

   C:\DSK\SIMULATE>dspa.exe (имя файла).asm

   DOS/4GW Professional Protected Mode Run-time Version 1.96

   Copyright (c) Rational Systems, Inc. 1990-1994

   TMS320C1x/C2x/C2xx/C5x COFF Assembler Version 6.60

   Copyright (c) 1987–1995 Texas Instruments Incorporated

   PASS 1

   PASS 2

   No Errors, No Warnings

4. Если синтаксических ошибок не обнаружено, то в учебных целях целесообразно внести в программу некоторую ошибочную информацию.

Например:

– “забыть” указать значение какой-либо переменной при определении данных;

– внести какую-либо синтаксическую ошибку в одну из команд, неправильно определив ее мнемокод;

– “забыть” определить используемую в программе переменную при определении данных;

5. Повторно оттранслировать полученную программу.

В данном случае, при обнаружении каких-либо ошибок на экране монитора появится предупреждение, примерный вид которого следующий:

Dos Navigator Version 1.42 Copyright (C) 1991,96 RIT Research Labs

C:\DSK\SIMULATE>dspa.exe sim.asm

DOS/4GW Professional Protected Mode Run-time Version 1.96

Copyright (c) Rational Systems, Inc. 1990–1994

TMS320C1x/C2x/C2xx/C5x COFF Assembler Version 6.60

Copyright (c) 1987–1995 Texas Instruments Incorporated

PASS

PASS 2

Y1 .word

"sim.asm", line 12: OPERAND MISSING

(в строке 12 текста программы не определено значение переменной Y1)

cPM #0

"sim.asm", line 27: INVALID OPCODE

(сообщение об ошибке в 27-й строке текста программы: среди команд ЦПОС TMS320C50 отсутствует команда CPM)

2 Errors, No Warnings (в программе обнаружены 2 ошибки)

Errors in source – Assembler Aborted

6. Прочитать содержимое файла-листинга ***(имя файла).lst***, созданного при трансляции исходного текста программы. В данном файле отражена более подробная информация об ошибках в программе.

7. Исправить ошибки в исходном тексте программы ***SIM*** и повторить операцию трансляции файла с текстом программы ***(имя файла).asm*** (см. п. 2).
8. Создать командный файл ***(имя файла).cmd,*** в котором указать распределение памяти процессора, используемое в программном имитаторе (см. п. 1.2 “Краткая теоретическая справка”).
9. Произвести операцию компоновки и получить выходной файл ***(имя файла).out*** и файл ***sim.map,*** набрав команду вида

   dsplnk.exe (имя файла).cmd

10. В случае, если командный файл составлен правильно, будет создан выходной файл ***(имя файла).out*** и файл ***(имя файла).map***. При этом на экране монитора появится следующая надпись:

C:\DSK\SIMULATE>dsplnk.exe sim.cmd

DOS/4GW Professional Protected Mode Run-time Version 1.96

Copyright (c) Rational Systems, Inc. 1990–1994

TMS320C1x/C2x/C2xx/C5x COFF Linker Version 6.60

Copyright (c) 1987–1995 Texas Instruments Incorporated

Следует иметь в виду, что при выполнении операции компоновки, сообщения об ошибках появляются только при нарушениях правил компоновки. Для контроля ошибочного размещения и соединения секций необходимо проанализировать таблицу компоновки – файл с расширением *.map, примерный вид которого с пояснениями приведен в п. 1.2.

11. Если ошибок при компоновке не обнаружено, то в учебных целях в командный файл следует внести некоторую ошибочную информацию. Например:

- изменить имя объектного файла;
- обнулить значение ячеек памяти, резервируемых для блока REGS;
- в блоке ***BLOCK0*** зарезервировать под память данных количество ячеек меньшее, чем требуется – не 10 ячеек, а 8.

При компоновке программы с помощью командного файла ***(имя файла).cmd*** на экране монитора в порядке внесения исправлений в командный файл должны появляться предупреждения, примерный вид которых представлен ниже:

TMS320C1x/C2x/C2xx/C5x COFF Linker Version 6.60

Copyright (c) 1987–1995 Texas Instruments Incorporated

>> : can't open file lab_sim.obj for input

(объектный файл с данным именем не существует)

TMS320C1x/C2x/C2xx/C5x COFF Linker Version 6.60

Copyright (c) 1987–1995 Texas Instruments Incorporated

>> (имя файла).cmd, line 13: zero or missing length for memory area REGS

(пропущено или равно нулю пространство памяти, зарезервированное для блока REGS)

TMS320C1x/C2x/C2xx/C5x COFF Linker Version 6.60

Copyright (c) 1987-1995 Texas Instruments Incorporated

>> cannot allocate .data in BLOCK0 (page 1)

>> errors in input – (имя файла).out not built

(невозможно разместить имеющееся количество данных в зарезервированных ячейках памяти программ, файл sim.out не создан)

11. После внесения исправлений в командный файл sim.cmd получить выходной файл ***(имя файла).out*** и проконтролировать инициализацию памяти процессора в файле ***(имя файла).map.***

В результате выполнения лабораторной работы должны быть получены следующие файлы:

- ***(имя файла).asm*** – файл, содержащий программу на языке ассемблера;
- ***(имя файла).lst*** – файл-листинг, позволяющий контролировать возможные ошибки в программе;
- ***(имя файла).obj*** – объектный файл, сформированный программой dspa.exe;
- ***(имя файла).cmd*** – командный файл компоновки;
- ***(имя файла).map*** – файл, позволяющий контролировать реально выполненное распределение памяти;
- ***(имя файла).out*** – сформированный выходной файл, непосредственно использующийся при работе с программой симулятора.

11. В соответствии с указаниями преподавателя подготовить программу с заданным модифицированным вариантом размещения секций программы в памяти процессора. Создать необходимый командный файл

компоновки. При создании этих файлов использовать имена, отличные от используемых ранее.

12. Произвести операцию компоновки и получить выходной файл ***(имя файла).out*** и файл ***имя файла).map,*** набрав команду вида

    dsplnk.exe (имя файла).cmd

Сравнить две полученные таблицы компоновки с различными распределениями секций программы.

1.5. Содержание отчета

Данная лабораторная работа носит ознакомительный характер и предназначена для практического освоения процесса создания загрузочных файлов для программы симулятора ЦПОС TMS320C50 и отладчика платы DSK. По результатам лабораторной работы отчет не составляется.

Контрольные вопросы

1. Поясните назначение операции трансляции программ. Какого рода предупреждения могут возникать при трансляции?
2. Поясните назначение, создаваемого транслятором листинг-файла ***(имя файла).lst***.
3. Каково назначение командного файла компоновки?
4. Поясните назначение операции компоновки программ. Какого рода предупреждения могут возникать при компоновке?
5. Поясните назначение файла ***(имя файла).map***.
6. Что изменится в командном файле при работе с платой DSK и с программой симулятора?
7. Каким образом задают области размещения выходных секций формируемого файла ***(имя файла).out*** в памяти процессора?

ЛАБОРАТОРНАЯ РАБОТА 2

ИЗУЧЕНИЕ СИМУЛЯТОРА ЦПОС TMS320C50 И ОСНОВНЫХ ПРИЕМОВ РАБОТЫ С НИМ

2.1. Цель работы и порядок выполнения

Целью работы является изучение пользовательского интерфейса и основных приемов работы с симулятором ЦПОС TMS320C50. Ознакомление с архитектурой процессора, картой распределения памяти, представлением данных в процессоре и основными приемами отладки и тестирования работы программы.

Работа выполняется в следующем порядке:

- изучение п. 2.2,
- подготовка задания в соответствии с указаниями п. 2.3,
- выполнение работы в соответствии с указаниями п. 2.4.

2.2 Краткая теоретическая справка

Симулятор (имитатор) предназначен для программной имитации работы ЦПОС TMS320C50 и отладки программ, предназначенных для выполнения на данном процессоре. Имитатор выполняет программы не в реальном масштабе времени. При выполнении программы в нем возможен контроль, а также модификация (изменение) состояния основных регистров процессора и содержимого памяти при тех или иных операциях.

Безусловным достоинством работы с симулятором является возможность имитировать работу цифрового сигнального процессора на любой IBM PC машине. Однако при реализации операций, при которых требуется обрабатывать большие объемы информации или анализировать изменения динамических процессов, происходящих в реальном масштабе времени, работа с симулятором становится крайне сложной и неэффективной. Подключение реальных внешних устройств к нему вообще невозможно.

Программа симулятора ***SIM5X*** обладает дружественным интерфейсом и позволяет вводить основные команды, управляющие работой, как в командной строке, так и посредством манипулятора “мышь”, используя экранное меню. Симулятор позволяет загружать и выполнять файлы, полученные в результате трансляции и компоновки программы, написанной на языке ассемблера, а также отлаживать программы на уровне языка **C**.

Графическая панель, формируемая программой, организована по оконному типу и дает возможность:

- загружать исполняемые программы и просматривать их дизассемблерную версию в окне ***DISASSEBLY***;
- выделять, перемещать и менять размеры окон;

- выполнять программы в пошаговом режиме и контролировать их выполнение, следя за изменениями, происходящими с содержимым регистров (окно ***CPU***) и областей памяти процессора (окно ***MEMORY***);

- задавать точки останова при непрерывном выполнении программы;
- сохранять содержимое различных областей памяти в отдельном файле, что позволяет сохранять и использовать для анализа данные, полученные в результате выполнения программ при работе с симулятором;

- вводить различные команды через командное меню, посредством клавиатуры (окно ***COMMAND***), или “мышью”;

- контролировать время выполнения программ.

После запуска программы симулятора на экране монитора появляется графическая панель, вид которой приведен на рис. 2.1. Панель отражает состояние всех регистров и памяти процессора, а также текст дизассемблированной программы.

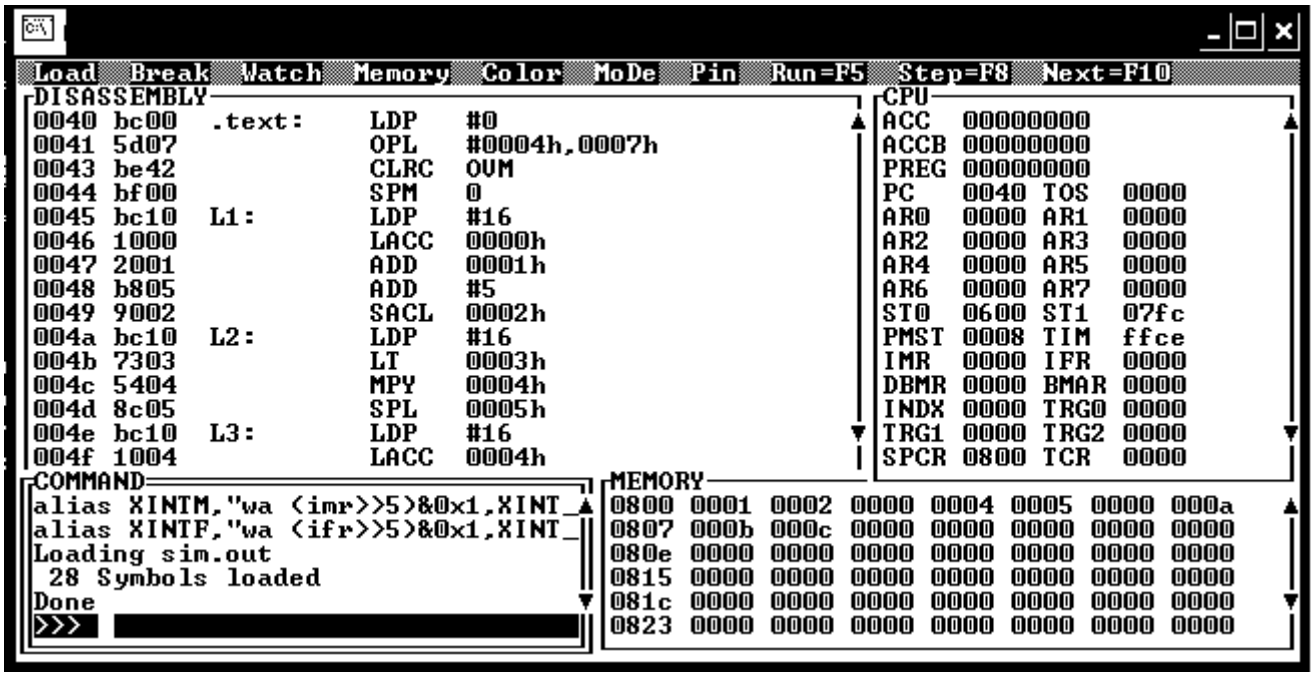

Рис. 2.1. Общий вид экрана при работе с симулятором

В качества примера, поясняющего общие принципы выполнения и отладки программ ЦОС при использовании симулятора, рассматривается программа ***SIM***, использованная в работе 1. Эта программа реализует простейшие операции при работе с ЦПОС TMS320C50.

2.3. Задание на самостоятельную подготовку

1. Изучите архитектуру ЦПОС TMS320C50, используя прил. А и материалы лекций.
2. Изучите основные регистры и карту распределения памяти процессора, используя материалы лекций и прил. А, В.

2.4. Выполнение лабораторной работы

2.4.1. Ознакомление с графическим интерфейсом пользователя

1. Скопируйте в свой рабочий директорий файлы:

- ***sim5x.exe*** – программа имитатора;

- ***alias.bat*** – файл с набором макрокоманд имитатора;
- ***siminit.cmd*** – управляющий файл для имитатора.

2. Запустите программу симулятора, набрав в командной строке ***DOS***:

sim5x.exe

После выполнения данной команды на экране монитора должно появиться основное окно графической панели (рис. 2.1). Изучите состав и расположение окон графического интерфейса, отражающих состояние основных элементов архитектуры процессора.

3. Выделите мышью пункт меню ***MEMORY,*** строка ***List***. При этом в окне команд будет выведена конфигурация областей памяти, доступной для отлаживаемой программы, с атрибутами, определяющими возможности их использования.

4. Выведите на экран в окне ***MEMORY*** содержимое памяти программ, начиная с адреса ***0000***, набрав в командном окне команду

mem 0x0000@prog

5. Выведите на экран в окне ***MEMORY*** содержимое памяти данных, расположенных на нулевой странице, начиная с адреса ***0000*** (т.е. участка памяти, зарезервированного для хранения состояния основных регистров процессора), набрав в командном окне команду

mem 0x0000@data

6. Выделите окно ***CPU,*** отражающее состояние основных регистров процессора. Нужное окно можно выделить, последовательно нажимая клавишу ***F6***, или “мышью”. Окно ***CPU*** можно выделить также командой ***win CPU.*** Подобную

команду можно использовать для любого окна, подставив его название, набранное прописными буквами.

7. Занесите в регистр ***AR0*** значение ***0ffffh.*** Для этого в окне ***CPU*** подведите “мышью” курсор к регистру ***AR0,*** нажмите левую кнопку “мыши”, введите новое значение и сохраните его, нажав клавишу ***Enter***. Проследите, как изменится при этом состояние памяти данных.

8. Повторите п. 7, изменяя состояния других регистров процессора.

9. Занесите в ячейку памяти ***0007h*** значение ***aaaah***, действуя в соответствии с указаниями п. 7.

10. Повторите п. 9, изменяя значения других ячеек памяти, начиная с адреса ***0007h*** и до адреса ***000dh***.

11. Сохраните измененное значение памяти данных, с адреса ***0007h*** до адреса ***000dh*** в файле ***DATA.OBJ.*** Для этого:

- в меню ***MEMORY*** выберите команду ***SAVE***;
- в открывшемся окне ***SAVE*** наберите информацию, как это показано на рис. 2.2, следует учитывать, что при сохранении памяти данных в графе ***PAGE*** должна стоять “1”, а при сохранении памяти программ –“0”.

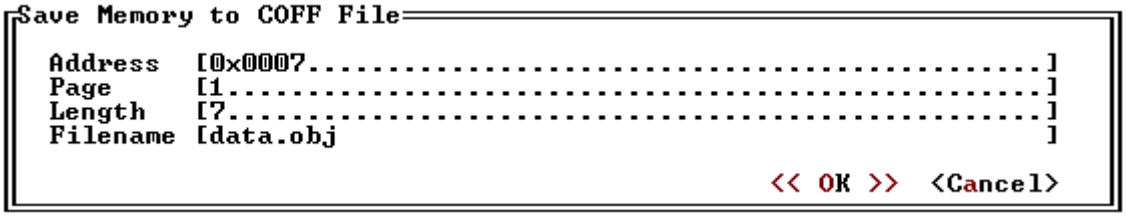

Рис. 2.2. Окно программы симулятора при сохранении пространства памяти

12. Выйдите из программы симулятора, набрав команду ***QUIT***, и просмотрите содержимое файла ***DATA.OBJ*** в текстовом редакторе ***DOS*** с представлением чисел в 16-ричном виде. Для этого последовательно нажмите клавиши ***F3*** и ***F4***. Убедитесь в том, что в файле ***DATA.OBJ*** сохранено содержимое ячеек выбранной области памяти данных (см. п. 11). Следует учитывать, что в файле ***DATA.OBJ*** первые 60 (3Bh) байт занимает служебная информация.

13. Войдите в программу симулятора и загрузите в нее файл с сохраненными изменениями содержимого памяти ***DATA.OBJ***, выбрав в меню ***LOAD*** команду ***LOAD***, как это показано на рис. 2.3.

Рис. 2.3. Окно программы симулятора при загрузке программы

14. Измените масштаб какого-либо окна, используя команду ***SIZE***, далее меняйте размеры окна клавишами управления курсором. Размер окна можно изменить также, используя "***мышь",*** ухватив правый нижний угол окна.

15. Переместите какое-либо окно. Это можно сделать, подведя курсор и "ухватив" ***"мышью***" верхний край рамки выбранного окна.

2.4.2. Загрузка и исполнение простейших программ

1. Загрузите файл с исполняемой программой***,*** предназначенной для изучения простейших операций, реализуемых на ЦПОС TMS320C50, который был подготовлен при выполнении работы 1.

Для загрузки исполняемой программы используйте меню ***LOAD*** и выберите в ней команду ***LOAD***. В открывшемся окне наберите имя файла исполняемой программы с расширением и нажмите клавишу Enter.

2. Используя ***"мышь"*** или клавишу ***F6,*** выделите окно дизассемблера и ознакомьтесь с текстом программы.

3. Выйдите из программы симулятора, набрав команду ***QUIT.***

4. Повторите операцию загрузки исполняемой программы другим способом. Для этого наберите в командной строке ***DOS***

SIM5X.EXE <имя программы>OUT

После загрузки имитатора с программой в окне ***MEMORY*** выводится содержимое ячеек памяти, где хранятся переменные ***X, Y, Z, X1, Y1, Z1, X2, Y2, Z2.*** (Карта памяти, показывающая местоположение данных ячеек, в памяти данных, представлена в лабораторной работе 1).

5. Откройте окно просмотра ***WATCH,*** позволяющее в отдельно создаваемом окне просматривать содержание не всех регистров процессора или данной области памяти, а лишь заданных. Выведете в окне ***WATCH*** значения переменных ***Z, X2, Y2, Z2,*** используемых в программе ***SIM.OUT*** для хранения промежуточных значений, а также значения регистров ***AR0, AR6*** и ***ACC***.

Для этого в пункте меню ***WATCH*** выберете команду ***ADD*** и наберите в открывшемся окне:

- для переменных:

  – адрес (в строке **Expression**);

  – имя переменной (в строке **Label**);

– формат вывода (в строке **Format**).

- для регистров – имя регистра .

Примерный вид набираемой информации для отражения представлен на рис. 2.4. и 2.5.

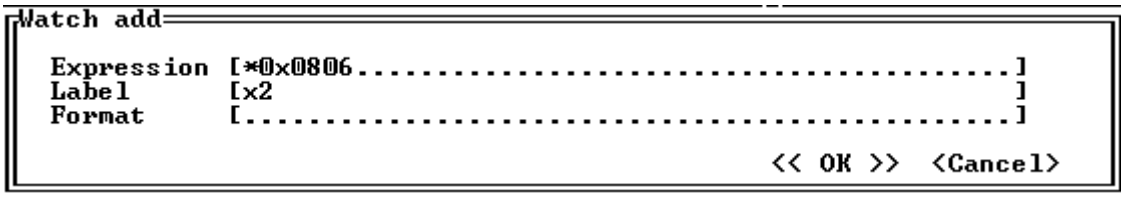

Рис. 2.4. Задание переменных для отображения в окне ***WATCH***

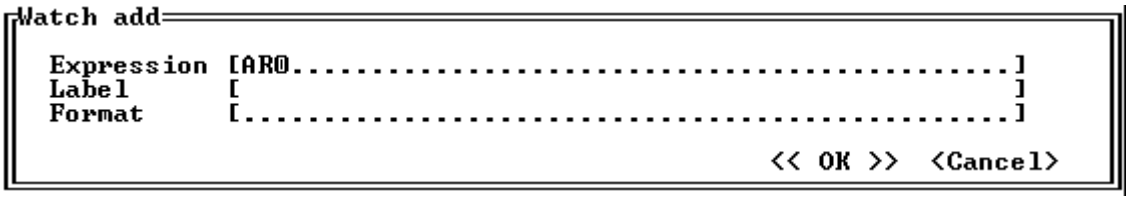

Рис. 2.5. Задание отображения содержимого регистра в окне ***WATCH***

Переменные и содержимое регистров могут быть отображены в 16-ричном (Hex) и десятичном форматах. Для выбора формата при задании переменных и регистров в строке Format необходимо указать соответствующий спецификатор: ***x*** для 16-ричного формата и ***d*** для десятичного. При отсутствии указания используется формат по умолчанию, которым является 16-ричный. На рис. 2.6 приведен пример вывода информации, причем для переменной ***x*** и содержимого ***AR0*** использован 16-ричный (строки 1, 3, 4) и десятичный (строки 2 и 5) форматы. Удаление переменной или регистра из окна ***WATCH*** производится выбором стоки ***Delete*** в пункте меню ***WATCH*** и указанием индекса (номера строки в окне) удаляемой переменной.

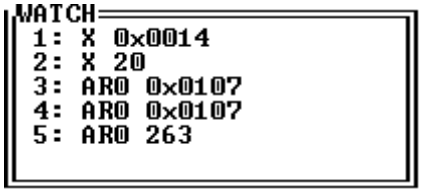

Рис. 2.6. Вывод информации в окне ***WATCH***

6. Для вывода числа в форме с плавающей запятой, нормированного к “1”, например, содержимого ACC, следует набрать команду вида

wa (float)((int)ACC)*((float)1/((float)32768*32768*2))

Аналогичное нормирование можно произвести для любой переменной.

7. В командной строке наберите команду ***RESTART***, осуществляющую операцию сброса и возврата в начало программы.

8. Выполните программу в пошаговом режиме, используя пункт меню ***STEP*** или ***F8***, и проследите за изменением содержимого регистров процессора и ячеек памяти данных, хранящих переменные, при выполнении каждой команды. Проследите за соответствием результатов ожидаемым.

9. Вернитесь в начало программы, набрав команду ***RESTART.*** При этом необходимо иметь в виду, что по этой команде изменяется только содержимое программного счетчика, а содержимое регистров и памяти не изменяется. Для возврата к состоянию с исходным содержимым необходимо выйти из программы и загрузиться вновь.

10. Выполните первые 10 команд программы. Для этого наберите команду ***STEP 10***. Проследите за изменением значений программного счетчика до и после выполнения команды ***STEP 10***, а также ячеек памяти данных и затем вернитесь в начало программы с помощью команды ***RESTART.***

11. Измените содержимое ячеек памяти ***X, Y, Z, X1, Y1, Z1, X2, Y2, Z2,*** занеся в них начальные значения.

12. Выполните часть программы с начала до метки ***L3***, набрав команду ***GO 0x0050*** (где ***0x0050*** адрес команды, соответствующей метке ***L3***), и проследите за произошедшими изменениями в окнах ***CPU, WATCH*** и ***MEMORY***.

Если в имитатор вместе с программой загружена таблица символов (символы-метки отражены в окне программы), то в команде ***GO*** можно указать метку ***GO L3.*** Проверьте эту возможность, выполнив команды до метки L5.

13. Вернитесь в начало программы командой ***RESTART***.

14. Запустите программу на выполнение командой ***RUN*** или клавишей ***F5.*** При этом программа будет выполняться до тех пор, пока не будет остановлена нажатием клавиши ***Esc.*** Команда, на которой остановится программа, будет случайной. При автоматическом выполнении программы содержимое выводимой на экран информации не изменяется, и произошедшие изменения можно увидеть только после остановки выполнения программы.

15. Для останова программы при автоматическом выполнении в любом нужном пользователю месте в ней могут быть установлены точки останова. При останове можно проконтролировать изменения переменных и содержимого регистров и памяти после выполнения определенных команд. Установите точку останова в нужном месте программы можно, поместив на нее курсор и нажав левую кнопку мыши. При этом выбранная строка программы в окне ***DISSASEMBLE*** будет подсвечена. В программе может быть установлено много точек останова. Выполнение программы между точками останова происходит

при выполнении команды ***RUN*** (или нажатии клавиши ***F5***). Для снятия точки останова нужно снова установить курсор в нужном месте программы и нажать левую кнопку мыши.

Установите в программе несколько точек останова и проверьте правильность выполнения программы.

16. При работе с программой симулятора можно измерить время выполнения программы в тактах (периодах) основной частоты работы процессора. Для этого необходимо вывести в окне ***WATCH*** значение регистра ***clk,*** как это показано на рис. 2.7. Регистр ***clk*** правильно отражает время выполнения программы между двумя точками останова. Для измерения времени выполнения нужного фрагмента программы необходимо в начале и в конце этого фрагмента установить точки останова, выполнить программу до начала фрагмента (команда ***RUN***) и затем выполнить команды программы до конца фрагмента (снова команда ***RUN***).

Измерьте время выполнения основной части программы.

Подобным образом можно проверять правильность выбора программы с точки зрения решения поставленной задачи.

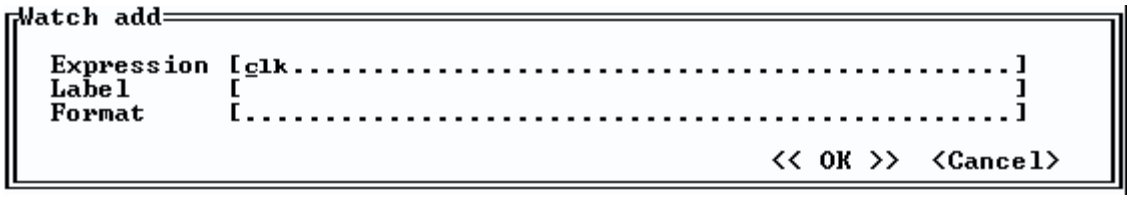

Рис. 2.7. Вид окна ***WATCH*** при вызове значения clk

17. Запустите программу на выполнение командой ***RUN*** и проследите за результатами работы.

18. Выйдете из программы симулятора, набрав в командной строке ***QUIT***.

2.4.3. Загрузка и исполнение варианта программы, выполненного самостоятельно

1. Загрузите вариант программы, подготовленной самостоятельно при выполнении работы 1 с измененным размещением секций и данных.

2. Проконтролируйте размещение секций и данных программы с помощью средств имитатора.

2.5. Содержание отчета

Данная лабораторная работа носит ознакомительный характер и предназначена для практического освоения программы симулятора при работе с ЦПОС

TMS320C50. По результатам лабораторной работы отчет не составляется. В конце работы проводится защита. При ответах на вопросы необходимо подтвердить навыки, полученные при работе с симулятором.

Контрольные вопросы

1. Поясните команды запуска программы на выполнение. Для чего может использоваться каждый из режимов запуска?

2. Поясните назначение основных регистров процессора и их место в памяти процессора.

3. Для чего используются точки останова при выполнении программы?

4. Чем отличается пошаговый режим выполнения программ от других режимов?

5. В каких форматах могут быть представлены числа в окнах имитатора?

6. Как сохранить содержимое выбранной области памяти?

7. Как в окне ***MEMORY*** вывести информацию о содержимом ячеек памяти данных, начиная с адреса 0200h?

8. Создайте окно просмотра ***WATCH*** и выведите в нем информацию о содержимом ячейки **Z2.**

9. Сохраните значения, содержащиеся в ячейках памяти данных, расположенных по адресам:

- начальный адрес 0F00h;
- конечный адрес 0F0Fh в файле ***DATA1.OBJ***.

10. Проделайте операцию сложения чисел 0FFh и 9. Выполните также эту операцию, используя программу ***SIM.***

11. Проделайте операцию перемножения чисел 8 и 3. Выполните также эту операцию, используя программу ***SIM.***

ЛАБОРАТОРНАЯ РАБОТА 3

Представление численных данных в процессорах с фиксированной запятой семейства TMS320C5x

3.1. Цель работы и порядок выполнения

Целью работы является изучение форматов представления целых, дробных, положительных и отрицательных чисел в цифровом процессоре обработки сигналов TMS320C50.

Работа выполняется в следующем порядке:

- изучение п. 3.2;
- подготовка задания в соответствии с указаниями п. 3.3;
- выполнение работы в соответствии с указаниями п. 3.4.

3.2. Краткая теоретическая справка

3.2.1. Двоичная система счисления

3.2.1.1. Представление чисел в двоичной системе счисления

Для того чтобы понять, как производятся арифметические операции в процессоре, необходимо изучить используемые в нем способы и форматы представления чисел.

Двоичная система счисления – это простейшая позиционная система, используемая в компьютерах и приборах цифровой техники. Она характеризуется следующими особенностями:

- каждый разряд может содержать либо 0 либо 1;
- количество разрядов определяет максимальную величину числа и диапазон представления чисел; это справедливо для любой позиционной системы счисления, но особенно важно в цифровой технике.

Пример:

$0110_2=(0 \cdot 2^3)+(1 \cdot 2^2)+(1 \cdot 2^1)+(0 \cdot 2^0)= (0 \cdot 8)+(1 \cdot 4)+(1 \cdot 2)+(0 \cdot 1)=6_{10}$;

$11110_2=(1 \cdot 2^4)+(1 \cdot 2^3)+(1 \cdot 2^2)+(1 \cdot 2^1)+(0 \cdot 2^0)= (1 \cdot 16)+(1 \cdot 8)+(1 \cdot 4)+(1 \cdot 2)+(0 \cdot 1)=30_{10}$.

3.2.1.2. Представление чисел в дополнительном двоичном коде

Числа, используемые на практике, могут иметь знак. Для представления чисел со знаком используются прямой, обратный и дополнительный коды. Прямой код предполагает наличие знакового (старшего) разряда.

Обратный код двоичного числа является промежуточным между прямым и дополнительным кодом.

Правило перевода числа из прямого кода в дополнительный:

- дополнительный код положительного числа совпадает с прямым;
- для перевода отрицательного числа необходимо:

– инвертировать все биты, т. е. заменить все “1” на “0” и все “0” на “1” во всех разрядах, кроме знакового;

– к полученному числу прибавить единицу.

Пример:

Исходное число со старшим знаковым разрядом (прямой код)	$0110_2 = 4 + 2 = 6_{10}$ (положительное)	$11110_2 = –16 + 8 + 4 + 2 = –14_{10}$ (отрицательное)
1. Инверсия	0110_2 (нет действия)	10001_2
2. Добавление единицы (результат в дополнительном коде)	0110_2 (нет действия)	$10010_2 = –14_{10}$

Особенности дополнительного кода:

- при использовании дополнительного кода операции сложения и вычитания над числами со знаками производятся, как над положительными беззнаковыми числами, т.е. операции со знаковыми разрядами производятся так же, как со значащими разрядами;
- правило перевода числа из дополнительного кода в прямой совпадает с приведенным выше правилом перевода из прямого в дополнительный.

Для чисел, представленных в дополнительном коде, при увеличении числа значащих разрядов, используемых для записи числа, необходимо копирование влево знакового разряда. Эта операция называется **расширением** знака. Например, число $–14_{10}$ при использовании семи разрядов, представленное в вышеприведенном примере в дополнительном коде пятью значащими разрядами 10010_2, будет иметь вид

1̲0010 → 111̲0010.

Последнее число легко проверить, преобразовав его в прямой код:

1110010 → инверсия: 1001101 → добавление 1: 1001110.

Старший разряд при расширении знака остается знаковым. Расширение знака позволяет записать число в регистр в случае, когда это число, представленное в дополнительном коде, имеет разрядность меньшую, чем регистр, в который его необходимо записать. Для этого необходимо производить следующие действия:

- загрузить исходное число в младшие разряды регистра;
- скопировать знаковый бит из исходного числа во все неиспользованные биты слева от исходного числа.

Рассмотрим запись двух величин из предыдущего примера в 8-битный регистр.

Пример:

Исходное число	$0110_2 = 4 + 2 = 6_{10}$	$10010_2 = –16 + 2 = –14_{10}$
1. Загрузка	0110_2	10010_2
2. Расширение знака	$00000110_2 = 4 + 2 = 6_{10}$	$11110010_2 = 2_{10} = –128+64+32+16+2 = –14_{10}$

3.2.1.3. Арифметические операции над числами в дополнительном коде

Как уже было отмечено, при использовании дополнительного кода операции сложения и вычитания чисел производятся, как над целыми беззнаковыми числами по правилам обычной арифметики. При этом знаковые разряды участвуют в операциях наравне со значащими. В результате получается алгебраическая сумма в дополнительном коде. Это позволяет производить операции, не обращая внимания на знаки чисел, а также использовать одни и те же устройства для операций как над числами со знаком, так и над беззнаковыми числами.

Пример. Операция сложения 6 + (–14) = –8

$$
\begin{array}{rl}
(6)_{\text{дк}} = & 000110 \\
(-14)_{\text{дк}} = & 110010 \\
(s)_{\text{дк}} = & 111000
\end{array}
\qquad
\begin{array}{r}
000110 \\
+\ 110010 \\
\hline
111000
\end{array}
$$

$$(s)_{\text{пк}} = 111000 \longrightarrow 100111 + 1 \longrightarrow \quad 101000 = (-8)_{10}$$

Пример. Операция вычитания $6 - (-14) = 20$

$$
\begin{array}{r}
000110 \\
-\ 110010 \\
\hline
010100 \longrightarrow \ 20_{10}
\end{array}
$$

При выполнении операции вычитания производится заем из мнимых (отсутствующих) разрядов. Результат операции вычитания получился положительный, и дополнительный код результата совпадает с прямым.

Алгоритм выполнения операции умножения над числами в дополнительном коде отличается от алгоритма умножения чисел в прямом коде. Один из возможных алгоритмов (алгоритм умножения, начиная с младших разрядов множителя, со сдвигом сумм частичных произведений вправо) выглядит следующим образом:

1. Исходное значение сумм частичных произведений принимается равным 0.
2. Если анализируемая цифра множителя равна 1, то к сумме частичных произведений прибавляется множимое. Если цифра равна 0, прибавление не производится.
3. Сумма частичных произведений сдвигается на один разряд вправо, при этом, если сумма отрицательна, осуществляется модифицированный сдвиг, т. е. сдвиг с расширением знака.
4. Последовательно выполняются пп. 2 и 3 для всех цифровых разрядов множителя, начиная с младшего.
5. Если множитель – положительное число, полученный результат представляет произведение в дополнительном коде. Если множитель отрицателен, то для получения произведения к результату прибавляется множимое с обратным знаком в дополнительном коде.

Пример: (– 4) * (– 3) = 12

$(-4)_{дк} = 1100$

$(-3)_{дк} = 1101$

	1100
*	1101
сумма	1100
модифицированный сдвиг	11100
	0000
сумма	11100
модифицированный сдвиг	111100
	1100
сумма	101100
модифицированный сдвиг	1101100
	1100
сумма	1001100
модифицированный сдвиг	11001100
добавление +4	0100
сумма = результат	00001100

При выполнении действий сложения частичных сумм переносы влево за пределы разрядной сетки игнорируются. Произведение имеет количество разрядов, равное сумме разрядов сомножителей, и, следовательно, два старших разряда – знаковые.

3.2.1.4. Представление дробных чисел в двоичном коде

Как и в представлении целых чисел, в представлении дробных чисел также используется дополнительный код. В отличие от представления целых чисел, где определено минимальное число 1, в дробных числах ограничено максимальное число. Обычно числа представляются в интервале от –1 до +1. При фиксировании разделительной запятой после знакового разряда количество разрядов определяет точность представления.

Пример: дробь в прямом коде

1	0	1	1
–1	1/2	1/4	1/8

$= -1 + 1/4 + 1/8 = -5/8$

При фиксировании запятой (или точки) после знакового (старшего) разряда перед старшим значащим разрядом перевод дробной части в дополнительный код производится по правилам, аналогичным правилам для целых чисел. Операции алгебраического сложения дробных чисел также аналогичны правилам для целых чисел.

3.2.2. Представление чисел в процессоре

3.2.1. Система команд и типы чисел процессора

Система команд процессоров с фиксированной запятой семейства TMS320, и в частности процессоров TMS320C5x, ориентирована на обработку двоичных чисел, представленных в дополнительном коде. Для операций алгебраического сложения (вычитания) это эквивалентно обработке беззнаковых целых чисел.

В процессоре не производится никаких преобразований кодов двоичных чисел. На пользователя (программиста) возлагается задача подачи на вход чисел в нужном формате и правильной трактовки результатов выполнения операций в процессоре. Следует отметить, что современные АЦП выдают на выходе численные значения отсчетов сигналов в дополнительном коде. В процессоре предусмотрен ряд опций и режимов работы, связанных с представлением чисел. Эти опции будут рассмотрены ниже.

При представлении чисел с фиксированной точкой в зависимости от положения точки могут рассматриваться либо целые, либо дробные числа. Обычно подразумевается, что точка находится или перед старшим значащим разрядом, или после младшего. Во втором случае в системе могут быть представлены только целые числа, в первом – дробные числа, по модулю меньше 1. Для 16-разрядных процессоров TMS320C5x формат представления дробных чисел определяется как формат Q15 (15 – число значащих разрядов справа от разделительной точки).

В процессорах TMS320C5x обычно предполагается обработка двух вариантов чисел:

1. Дробные числа в формате Q15. Таковыми являются, в частности, отсчеты входных сигналов, подаваемые на вход процессора с АЦП (по модулю меньше 1).
2. Целые числа. Таковыми могут являться значения счетчиков повторений, номера отсчетов, адреса в памяти и т. д.

Следует отметить, что промежуточные значения вычисляемых дробных величин на выходе умножителя и АЛУ могут быть представлены и в других форматах (в частности, Q31).

В процессорах с фиксированной точкой возможна обработка дробных чисел в формате с плавающей точкой, но только с помощью программирования (на основе составления соответствующей программы преобразования и обработки).

При выполнении вычислений с использованием дробных чисел 32-разрядный аккумулятор позволяет увеличить точность вычислений и уменьшить накопление ошибки. Ошибка возникает из-за того, что погрешность представления дробных чисел в двоичном коде всегда составляет ± половина значения младшего разряда. Следовательно, при сложении двух двоичных чисел возникает ошибка ± значение младшего разряда. Например, при последовательном выполнении 256 сложений, ошибка составляет 8 младших разрядов. Применение 32-разрядного аккумулятора позволяет уменьшить

погрешность вычисления промежуточных результатов при накоплении произведений. При сохранении результата в 16-разрядной ячейке памяти используются старшие разряды аккумулятора.

3.2.2.2. Операции алгебраического сложения/вычитания

Эти операции выполняются в АЛУ. Один из операндов при выполнении операции берется из аккумулятора ACC, результат также помещается в ACC. При этом надо иметь в виду следующее. ACC и АЛУ являются 32-разрядными устройствами, а операнды, поступающие на вход процессора и хранимые в ЗУ, являются 16-разрядными. Загружать числа в ACC можно либо в его старшее слово (старшие 16 разрядов), либо в младшее слово, либо с каким-то сдвигом. Выполнять операции сложения/вычитания можно либо со старшим словом, либо с младшим словом, либо со всем содержимым ACC (в зависимости от используемой команды), и результат выполнения операции также будет находиться в соответствующем месте ACC. При сохранении результата операции в памяти необходимо использовать команды сохранения старшего (SACH) или младшего слова ACC (SACL), а при работе с повышенной точностью – все 32 разряда (в двух ячейках ЗУ) двумя командами (SACH и SACL).

Как уже было отмечено выше, в процессоре предполагается использование либо целых чисел, либо дробных чисел в форматах Q15 и Q31/Q30.

Команды процессора выполняют операции над всеми типами чисел одинаково. Правильное формирование исходных операндов и трактовка результата полностью лежат на пользователе (программисте). Ошибки могут возникнуть при загрузке операндов в младшее слово ACC или при выполнении операций, связанных с операндами, расположенными в младшем слове.

Пример: $0 - (-5) = 5$ (должно быть);

$(-5)_{дк} = fffb_{16}$ (в ячейке ЗУ).

Вариант 1 выполнения операции:

```
  0 0 0 0 0 0 0 0
–           f f f b
  ---------------
  f f f f 0 0 0 5  ──────►  f f f b16
```

Результат вычитания при переводе в прямой код дает отрицательное число $fffb_{16}$.

Вариант 2 выполнения операции:

```
  0 0 0 0 0 0 0 0
–
  f f f f f f f b
  ---------------
  0 0 0 0 0 0 0 5  ──────►  + 5
```

Второй вариант выполнения, который дает правильный результат, отличается тем, что при выполнении операции произведено расширение знака операнда “–5”. Процессоры TMS320 с фиксированной запятой дают возможность работы в

режимах с расширением знака и без него. Выбор нужного режима производится пользователем.

3.2.2.3. Режим расширения знака процессора

Режим работы определяется состоянием бита **SXM** (Sign Extension Mode), который расположен в регистре состояния процессора. Он показывает, использовать расширение знака или нет.

Установка бита выполняется командами:

```
SETC    SXM     ; установка бита SXM – АЛУ оперирует
                ; с числами в дополнительном коде
CLRC    SXM     ; сброс бита SXM – АЛУ оперирует
                ; с беззнаковыми числами
```

После общего сброса процессора бит **SXM** установлен в 1.

3.2.2.4. Операции умножения

Операция умножения выполняется в аппаратно реализованном специализированном узле, который называется умножителем. Умножитель реализует алгоритм умножения двоичных чисел в дополнительном коде.

В результате умножения 16-разрядных чисел получается 32-разрядный результат, два старших разряда которого, как показано в п. 3.2.1.3, являются знаковыми.

При умножении дробных чисел в формате Q15 старшее слово регистра P, в котором находится результат, является его округленным значением в формате Q14 (с двумя знаковыми разрядами) с отброшенными младшими значащими разрядами.

При умножении чисел результат содержится во всех 32 разрядах. Если необходимо ограничиться при умножении целых чисел одним словом (например, для запоминания в ячейке ЗУ), то умножать нужно числа, величина которых не превышает 2^8, и результат будет находиться в младшем слове регистра P.

Таким образом, в зависимости от того, какие числа перемножаются, результат может находиться либо в старшем, либо в младшем слове регистра P.

Вторым “тонким местом” при использовании результата умножения являются “лишние” знаковые разряды. Здесь возможны два варианта решения.

1. Результаты умножения накапливаются в АЛУ с лишними знаковыми разрядами, а ликвидировать лишние знаки можно при сохранении конечного результата в памяти. Если необходимо сохранить в памяти старшее слово

аккумулятора, его содержимое необходимо сдвинуть на один бит влево. Это может быть осуществлено за два цикла как, показано в следующем примере.

Пример:

```
; A*B=C

LT      A       ; A и B в формате Q15
MPY     B       ; P=A*B : формат Q30
PAC             ; ACC=A*B в формате Q30

SFL             ; ACC=A*B в формате Q31

SACH    C       ; C=A*B в формате Q15
```

Результат может быть сохранен быстрее, за 1 цикл, если использовать команду **SACH** со сдвигом.

Пример:

```
; A*B=C

LT      A       ; A и B в формате Q15
MPY     B       ; P=A*B : формат Q30
PAC             ; ACC=A*B в формате Q30

SACH    C,1     ; C=A*B в формате Q15
```

2. Ликвидация лишних знаковых разрядов путем сдвига содержимого регистра P влево. Этот сдвиг может осуществить сдвигатель P-регистра. Сдвигатель P-регистра обозначается как “P-scaler” и располагается между P-регистром и входным мультиплексором АЛУ (см. прил. А). Этот сдвигатель может передавать содержимое P-регистра без сдвига, со сдвигом влево на 1 или 4 разряда или со сдвигом вправо на 6 разрядов. Сдвигатель управляется двумя разрядами PM, расположенными в регистре управления ST1. Содержимое PM может быть изменено командой **SPM** (Set PM bits). Зависимость между величиной, записанной в PM, и результирующим сдвигом приведена в следующей таблице.

PM	Сдвиг
0	Нет сдвига
1	Сдвиг влево на 1

2	Сдвиг влево на 4
3	Сдвиг вправо на 6

Если установить режим сдвига на 1 бит влево, то при сохранении в аккумуляторе содержимого P-регистра формат представления чисел в аккумуляторе будет Q31 (при умножении чисел в формате Q15).

Пример:

```
SPM     1       ; Режим сдвига – на 1 бит влево

;A*B=C

LT      A       ; A и B в формате Q15
MPY     B       ; P=A*B : в формате Q30
PAC             ; Acc=A*B в формате Q31
SACH    C       ; C=A*B в формате Q15
```

После общего сброса PM устанавливается в 0.

В следующем примере реализовано вычисление выражения $y = m \cdot x + b$ в двух вариантах. Первый вариант реализует его в формате Q30, а второй – в формате Q31. Обратите внимание на отличия в прибавлении величины **b** к аккумулятору.

Пример:

```
; Q30 формат

SPM     0       ; режим Q30
LT      X
MPY     M       ; P=m*x : Q30
PAC             ; Acc=m*x : Q30
ADD     b,15    ; Acc=m*x+b : Q31
```

```
        SACH    y,1     ; y=m*x+b : Q15
```

;Q31 формат

```
        SPM     1       ; режим Q31
        LT      X
        MPY     M       ; P=m*x : Q30
        PAC             ; Acc=m*x : Q31
        ADD     b,16    ; Acc=m*x+b : Q31
        SACH    Y       ; y=m*x+b : Q15
```

Режим сдвига на 4 бита влево позволяет получить результат в формате Q31, при использовании команды MPYK умножения на 13-разрядную константу (в арифметике Q12).

3.2.3 Эффект переполнения аккумулятора

Явление переполнения происходит при выходе значащих разрядов за пределы разрядной сетки.

Двоичное представление дробных чисел в дополнительном коде имеет следующие особенности: если к (+1) в двоичном коде прибавить 1/32767, получится –1, т. е. число кардинально изменяется: вместо большого положительного становится большим отрицательным.

3.2.3.1. Проверка переполнения

В процессоре TMS320C50 бит контроля переполнения OV (Overflow) располагается в регистре состояния и устанавливается в случае, когда величина в аккумуляторе переходит границу между большими положительными и отрицательными числами, а также в случае, если осуществлен перенос в 33-й несуществующий бит. Некоторые подробности о бите переполнения приведены ниже:

- Переполнение удобно проверять с помощью команд условных переходов, таких как BCND <pma>,OV.
- Для достоверности результатов необходимо сбрасывать бит OV перед началом новых вычислений.

- Бит OV сбрасывается в 0 при аппаратном сбросе процессора.

3.2.3.1. Режим работы с переполнением

В процессорах TMS320 с фиксированной запятой предусмотрен режим работы с переполнением. В этом режиме при выходе положительного числа за пределы разрядной сетки устанавливается наибольшее возможное положительное число, а при выходе отрицательного числа за пределы разрядной сетки устанавливается наибольшее по модулю отрицательное число. Таким образом, при работе в этом режиме процесс переполнения напоминает то, что происходит в аналоговых устройствах при ограничении сигнала.

Режим работы с переполнением или без определяется значением бита OVM регистра состояния ST1. При значении 1 происходит работа в режиме с переполнением, при значении 0 – без переполнения.

Значение бита OVM, равное 0, может быть установлено командой ROVM или CLRC OVM, значение 1 – командой SOVM или SETC OVM.

3.2.4. Запись дробных чисел в процессорах TMS320 в формате Q15

Соответствие исходных дробных чисел и двоичных чисел в процессоре показано на следующей числовой линии.

Чтобы записать исходное дробное число в формате Q15 процессора его необходимо умножить на 32768, отбросить дробную часть и перевести в 16-ричную систему (в дополнительном коде, если исходная дробь отрицательна).

Пример:

Представить 0.62: 0.62*32768=20316. В 16-ричной системе: 4F5C.

Представить 0.1405: 0.1405*32768= 4603. В 16-ричной системе: 11F9.

Ассемблер позволяет производить эти вычисления при выполнении трансляции программы.

Пример организации загрузки дробных констант в память:

```
A       .word       32786*5/10          ; A= 0.5
B       .word       32768*25/100        ; B=0.25
```

Для перевода результата вычислений, произведенных в процессоре, из формата Q15 в обычный вид необходимо проделать обратные преобразования:

- перевести число из 16-ричной системы в десятичную;
- разделить полученное число на 32768.

3.2.5. Стандартная конфигурация регистров состояния процессора

Ниже приведен пример последовательности инструкций, которые необходимо включить в процедуру инициализации процессора.

```
SETC        SXM     ; разрешить использовать дополнительный код

CLRC        OVM     ; разрешить переполнение

SPM         1       ; осуществлять сдвиг на 1 бит влево

                    ; при копировании P-регистра в аккумулятор
```

3.3. Задание на самостоятельную подготовку

1. Изучите лекционный материал и материал теоретической справки (п. 3.2).
2. Изучите программу lab3.asm, которая приведена ниже. Эта программа будет выполняться в лаборатории.
3. Рассчитайте вручную точные значения величин Y и X, вычисляемых при выполнении программы и всех получаемых при этом промежуточных значений.

```
.version 50
.title "lab3_asm"

* Изучение представления чисел и влияния опций

* Вычисление Y=(I*I)+(IL*IL)+(ILL*ILL) для целых чисел

* Вычисление X=(A*B)+(C*D)+(E*F) для дробных чисел
```

```
        .global M1,INT,FLOAT1,FLOAT2,FLOAT3,FLOAT4
        .global FLOAT5,FLOAT6,SXM,HALT,Q30,Q31,X,Y


        .bss    A,13        ; резервирование памяяти для
B       .set    A+1         ; констант и переменных
C       .set    B+1
D       .set    C+1
E       .set    D+1
F       .set    E+1
X       .set    F+1
I       .set    X+2
IL      .set    I+1
ILL     .set    IL+1
Y       .set    ILL+1

*  Таблица значений констант, размещенных в памяти программ;

*  32768 соответствует единице

        .data
TABLE   .word   32768*9/10, 32768*8/10, 32768*7/10  ; A=0.9, B=0.8, C=0.7
        .word   32768*6/10, 32768*–1, 32768*4/10    ; D=0.6, E=–1.0, F=0.4
        .word   0, 0, 5, 100                        ; X=0, 0, I=5, IL=100
        .word   1000,0,0                            ; ILL=1000, Y=0, Y+1=0
        .sect   “Vector”
RESET           B                                   M1
        .text
M1:



* Перепись констант из памяти программ в память данных

* в зарезервированное место

LAR     AR1,#A
MAR     *,AR1
LACC    #TABLE
RPT     #12
TBLR    *+

**************************

* Работа с целыми числами в младших разрядах аккумулятора

* Вычисление      Y=(I*I)+(IL*IL)+(ILL*ILL)
```

INT:

```
LDP     #A          ; указатель страницы памяти на размещение A
ZAC                 ; (ACC)=0
SACL    Y+1         ; сохранение младших разрядов ACC – ACCL
LT      I
MPY     I           ; (P)=I*I
LTA     IL          ; (ACC)=I*I
SACL    Y+1         ; сохранение ACCL
MPY     IL          ; (P)=IL*IL
LTA     ILL         ; (ACC)=I*I+IL*IL
SACL    Y+1         ; сохранение  ACCL
MPY     ILL         ; ILL*ILL
APAC                ; (ACC)=(I*I)+(IL*IL)+(ILL*ILL)
SACL    Y+1         ; сохранение младшего слова ACCL
SACH    Y           ; сохранение старшего слова ACCH
```

****

* Работа с дробными числами

* Вычисление X=(A*B)+(C*D)+(E*F)

FLOAT1:

* Влияние опции режима переполнения OVM

* Влияние сдвига результата вычисления при передаче его из ACC в ЗУ

* Отсутствие сдвига

```
        LDP     #A
        SPM     0       ; PM = 0
        CLRC    OVM     ; OVM = 0
        CALL    Q31     ; вызов подпрограммы вычисления X
FLOAT2:
        SETC    OVM     ; PM = 0, OVM = 1
        CALL    Q31     ; вызов п/программы вычисления X
FLOAT3:
```

* Влияние сдвига результата вычисления при передаче его из ACC в ЗУ:

* Наличие сдвига

```
        LDP     #A
        SPM     0       ; PM = 0
        CLRC    OVM     ; OVM  0
        CALL    Q30     ; вызов п/программы вычисления X
```

```
FLOAT4:
        SETC    OVM     ; PM = 0, OVM = 1
        CALL    Q30     ; вызов п/программы вычисления X
FLOAT5:
```

* Влияние сдвига результата умножения при передаче

* из регистра P в АЛУ

* при отсутствии сдвига общего результата при передаче из ACC в ЗУ

```
        SPM     1       ; PM = 1
        CLRC    OVM     ; OVM = 0
        CALL    Q31     ; вызов подпрограммы вычисления X
FLOAT6:
        SETC    OVM     ; OVM = 1
        CALL    Q31     ; вызов подпрограммы вычисления X
SXM:
```

* Влияние опции SXM – режим расширения знака

* SXM влияет на выполнение операций в младшем

* слове ACC. При SXM=0 числа считаются беззнаковыми,

* при SXM=1 числа считаются знаковыми в дополнительном коде.

* SXM=0

```
        CLRC    SXM
        ZAC
        SUB     #5      ; (ACC)=0 – 5
        ZAC
        UB      #(–5)   ; (ACC) = 0 – ( –5)
        LACK    #(–5)   ; загрузка в ACC числа
        ADD     #6      ; (ACC)=–5 + 6
        LACK    #(–5)   ; загрузка в ACC числа
        SUB     #(–6)   ; (ACC)= – 5 – (–6)
*  SXM=1
        SETC    SXM
        ZAC
        SUB     #5      ; (ACC)=0–5
        ZAC
        SUB     #(–5)   ; (ACC)=0 – (–5)
        LACK    #(–5)   ; загрузка в ACC числа
        ADD     #6      ; (ACC)= – 5 + (6)
        LACK    #(–5)   ; загрузка в ACC числа
```

```
        SUB     #(–6)       ; (ACC)= –5 – (–6)
HALT:
        NOP
```

* Подпрограмма вычисления X со свигом на 1 разряд

* при передаче из ACC в ЗУ

```
Q30:
        ZAC                 ; (ACC)=0
        SACH    X,1         ; сохранение старшего слова ACC со сдвигом на 1  разряд
        LT      A           ; (T)=A
        MPY     B           ; (P)=A*B
        LTA     C           ; (T)=C  (ACC)=A*B
        SACH    X,1         ; сохранение старшего слова ACC со сдвигом на 1  разряд
        MPY     D           ; (P)=C*D
        LTA     E           ; (T)=E   (ACC)=A*B+C*D
        SACH    X,1         ; сохранение старшего слова ACC со сдвигом на 1 разряд
        MPY     F           ; (P)=E*F
        APAC                ; (ACC)=A*B+C*D+E*F
        SACH    X,1         ; сохранение старшего слова ACC со сдвигом на 1 разряд
        RET
```

* Подпрограмма вычисления X без сдвига на 1 разряд при передаче

* из ACC в ЗУ

```
Q31:
        ZAC
        SACH    X
        LT      A
        MPY     B
        LTA     C
        SACH    X
        MPY     D
        LTA     E
        SACH    X
        MPY     E
        APAC
        SACH    X
        RET
        .end
```

3.4. Выполнение лабораторной работы

1. Скопируйте в свой рабочий директорий файлы:

- ***sim5x.exe*** – программа имитатора;
- ***alias.bat*** – файл с набором макрокоманд имитатора;
- ***siminit.cmd*** – управляющий файл для имитатора;
- ***lab3.asm*** – исходный текст программы лабораторной работы;
- ***lab3.cmd*** – командный файл компоновки;
- ***lab3.txt –*** текстовый файл***.***

2. Выполните трансляцию и компоновку программы lab3.asm.

Для трансляции наберите в командной строке команду

dspa.exe –lsc lab3.asm

Для компоновки наберите в командной строке команду

dsplnk.exe lab3.cmd

3. Запустите имитатор процессора, набрав команду ***sim5x.exe***

При этом автоматически загрузится программа лабораторной работы ***lab3.out***, файл с макрокомандами ***alias.bat*** и текстовый файл ***lab3.txt*** с указанием некоторых команд, используемых в лабораторной работе.

4. Выполните работу, следуя указаниям, приведенным ниже.

3.4.1. Указания к выполнению лабораторной работы

Пункты указаний выделяются метками, одинаковыми с метками разделов программы, к которым данные пункты относятся. В тексте указаний частично повторяются команды программы. Изучение проводится на примере вычислений следующих выражений:

Y = (I*I) + (IL*IL) + (ILL*ILL) для целых чисел,

X = (A*B) + (C*D) + (E*F).

Выражения вычисляются путем последовательного накопления входящих в них произведений. Перед переходом к выполнению работы необходимо выполнить вычисление точных значений выражений. Используемые значения коэффициентов приведены ниже в таблице TABLE. При использовании

дробных чисел их значения нормируются к единице, которой соответствует величина 32768.

1. Размещение переменных и констант в памяти.

```
        .bss    A,13
B       .set    A+1
C       .set    B+1
D       .set    C+1
E       .set    D+1
F       .set    E+1
X       .set    F+1
I       .set    X+2
IL      .set    I+1
ILL     .set    IL+1
Y       .set    ILL+1
```

2. * Таблица значений констант, размещенных в памяти программ;

* 32768 соответствует единице

```
TABLE   .word   32768*9/10, 32768*8/10, 32768*7/10    ; A=0.9, B=0.8, C=0.7
        .word   32768*6/10, 32768*–1, 32768*4/10      ; D=0.6, E= –1.0, F=0.4
        .word   32768*0,0,5,100                       ; X=0, XA=0, I=5, IL=100
        .word   1000,0,0                              ; ILL=1000, Y=0, Y+1=0
```

3. **M1:** Перепись констант из памяти программ в память данных.

Наберите команду ***mem 0x200***

Выполните команду ***Go INT***. При этом должна быть выполнена часть программы до метки **INT** и в памяти данных появятся переписанные константы из памяти программ.

4. **INT:** Работа с целыми числами в младших разрядах аккумулятора. Вычисление выражения Y = (I*I) + (IL*IL) + (ILL*ILL).

Выполните ряд команд имитатора для отображения исследуемых величин в окне ***WATCH***:

- команду **DP** – отображение указателя страницы памяти данных;
- команду **ACCL** – отображение содержимого ячейки Y+1 в десятичной системе;

- команду **ACCI** – отображение содержимого ACC в десятичной системе.

Выполните фрагмент программы до **FLOAT1** по шагам.

Сравните результаты вычисления всех частичных сумм и последнего результата с точными значениями, полученными при ручном расчете. Обратите внимание, что ошибка появляется при переполнении младшего слова ACC после прибавления третьего произведения.

5. **FLOAT1:** Работа с дробными числами.

Вычисление выражения X = (A*B) + (C*D) + (E*F).

Влияние следующих установок и опций:

OVM – работа в режиме переполнения, сдвиг содержимого ACC (результата вычисления) при записи его в память.

Удалите из окна ***WATCH*** значения **ACCL**, **ACCI** (меню ***WATCH,*** пункт ***Delete***).

Выполните ряд команд имитатора для отображения исследуемых величин в окне ***WATCH***:

- команду **PM** – вывод значения опции PM для отображения величины сдвига при передаче содержимого регистра P в ACC;
- команду **OVM** – вывод значения опции OVM – указание на работу в режиме переполнения;
- команду **ACCF** – вывод содержимого ACC, как нормированного дробного числа;
- команду **ACCX** – вывод числа X, записанного в память, как нормированного дробного числа.

Выполните по шагам фрагмент программы до метки **SXM**. Результаты выполнения команд процессора (результаты вычислений) занесите в таблицу типа табл. 3.1. В нее заносятся значения всех частичных сумм и последнего результата в зависимости от опций и сдвига; сравните результаты с точными расчетами. Значения частичных сумм и последнего результата видны в окне ***WATCH*** (значения **ACCF**, **ACCX**).

5.1. Отсутствие сдвига содержимого ACC.

```
        LDP     #A
        SPM     0           ; PM = 0
        ROVM                ; OVM = 0
        CALL    Q31
FLOAT2:
        SOVM                ; PM = 0, OVM = 1
```

```
            CALL        Q31
```

5.2. Наличие сдвига содержимого ACC.

```
FLOAT3:     LDP         #A
            SPM         0           ; PM = 0
            ROVM                    ; OVM  0
            ALL         Q30
FLOAT4:
            SOVM                    ; PM = 0, OVM = 1
            CALL        Q30
```

5.3. Влияние опции PM – сдвига результата умножения – (содержимого регистра P) при передаче в АЛУ. Сдвиг содержимого ACC при записи в память отсутствует.

```
FLOAT5:     SPM         1           ; PM = 1
            ROVM                    ; OVM = 0
            CALL        Q31
FLOAT6:
            SOVM                    ; OVM = 1
            CALL        Q31
```

6. **SXM:** Влияние опции SXM – режим расширения знака. SXM влияет на выполнение операций в младшем слове ACC. При SXM = 0 числа считаются беззнаковыми, при SXM = 1 числа считаются знаковыми в дополнительном коде.

Удалите сообщения ACCF, OVM, PM, ACCX из окна ***WATCH*** (меню ***WATCH,*** пункт ***Delete***). Выполните команду **SXM** – вывод признака SXM. Наберите команду **ACCI** – вывод содержимого ACC в десятичной системе. Выполните программу по шагам до метки **HALT** и заполните табл. 3.2 результатов в зависимости от выполняемых арифметических операций **ADD**, **SUB** и значения **SXM**.

```
*  SXM=0
            RSXM
            ZAC
            SUB         #5          ; (ACC) = 0 – 5
            ZAC
            SUB         #(–5)       ; (ACC) = 0 – (–5)
            LACK        #(–5)
```

```
                ADD     #6          ; (ACC) = – 5 + 6
                LACK    #(–5)
                SUB     #(–6)       ; (ACC) = – 5 – (–6)
*  SXM=1
                SSXM
                ZAC
                SUB     #5          ; (ACC) = 0 – 5
                ZAC
                SUB     #(–5)       ; (ACC) = 0 – (–5)
                LACK    #(–5)
                ADD     #6          ; (ACC) = – 5 + (6)
                LACK    #(–5)
                SUB     #( –6)      ; (ACC) = – 5 – (–6)
HALT:
                NOP
                .end
```

Таблица 3.1

PM	OVM	Сдвиг при записи в память	Промежуточные и последние результаты			Примечания
			точные	ACC (ACCF)	память (ACCX)	
0	0	0	0.72 1.14 0.74			
0	1	0				
0	0	1				
0	1	1				
1	0	0				
1	1	0				

Таблица 3.2

SXM	Операция	Результат

3.5. Содержание отчета

1. Текст программы с подробными комментариями.

2. Табл. 3.1 и 3.2, содержащие состояния регистров PM, SXM, OVM и результаты работы программы.

3. Выводы по результатам сравнения точных значений, полученных при ручном расчете и значений, полученных в результате выполнения программы.

Контрольные вопросы

1. Поясните последовательность перевода двоичного положительного числа в двоичное отрицательное число.
2. Поясните запись “короткого” числа в “длинный” регистр.
3. Переведите в двоичный код следующие числа:

3, –3, 5, –5, 30, –30, 100, 64, –65, –87.

4. Почему после выполнения умножения необходимо сохранять старшее слово аккумулятора?
5. Зачем при сохранении старшего слова аккумулятора применяется сдвиг влево на 1 разряд?
6. Что означает “формат Q15”?
7. Чем определяется точность представления целых двоичных чисел?
8. Чем определяется точность представления дробных двоичных чисел?
9. Что произойдет, если к 1 в формате Q15 прибавить 1?

10. Что произойдет, если из 0 в формате Q15 вычесть 1?

ЛАБОРАТОРНАЯ РАБОТА 4

Изучение методов адресации процессора TMS320C50

4.1. Цель работы и порядок выполнения

Целью работы является изучение основных методов адресации данных, используемых в процессоре TMS320C50.

Работа выполняется в следующем порядке:

- изучение п. 4.2;
- подготовка задания в соответствии с указаниями п. 4.3;
- выполнение работы в соответствии с указаниями п. 4.4.

4.2. Краткая теоретическая справка

В процессорах семейства TMS320C5x возможны четыре основных режима адресации памяти данных:

- Прямая адресация.
- Косвенная адресация.
- Непосредственная адресация.
- Циклическая адресация.

Используется также бит-реверсивная адресация, являющаяся разновидностью косвенной.

4.2.1. Прямая адресация

Вся область памяти данных процессора объемом 64К разделена на 512 страниц размером 128 16-разрядных слов каждая. В режиме прямой адресации, команда содержит только младшие 7 бит адреса ячейки памяти. Это поле при обработке адреса дополняется 9 битами регистра указателя страницы (*DP*) до полного 16-разрядного адреса. Таким образом, регистр *DP* указывает на одну из 512 возможных страниц памяти размером 128 слов, а 7-битовый адрес в слове команды указывает на определенную ячейку памяти на данной странице памяти. Регистр *DP* может быть загружен при помощи команд **LDP** (*load data memory page pointer*) или **LST #0** (*load status register STO*). Схема образования полного 16-разрядного адреса при прямой адресации приведена на рис. 4.1.

Примечание. Указатель страницы не инициализируется при сбросе и, следовательно, не задан после включения питания. Однако средства разработки *C5x* устанавливают по умолчанию многие параметры, в том числе и регистр указателя страницы. Вследствие этого программы, не инициализирующие указатель страницы, могут работать неверно. Следовательно, используемая программа должна сама инициализировать указатель страницы памяти.

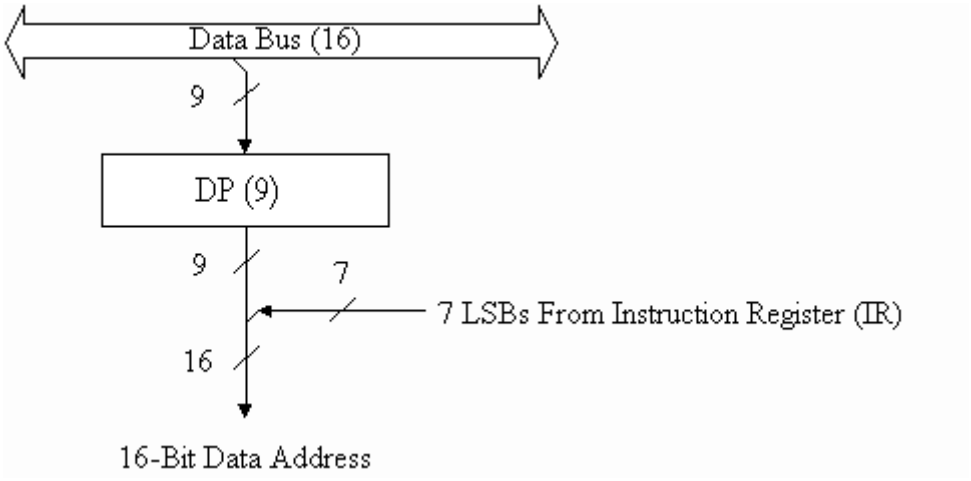

Рис. 4.1. Диаграмма прямой адресации

Пример прямой адресации:

```
LDP #2          ; загрузить в DP число 2, после чего текущей будет
                ; вторая страница
ADD 9h, 5       ; прибавить к содержимому аккумулятора содержимое
                ; ячейки памяти с адресом 9h на второй странице
                ; (полный адрес 265 или 109h), сдвинутое
                ; на 5 разрядов влево
LDP #5          ; Загрузить в DP число 5, после чего текущей будет
                ; пятая страница
ADD 9h, 5       ; Прибавить к содержимому аккумулятора содержимое
                ; ячейки памяти с адресом 9h на пятой странице,
                ; сдвинутое на 5 разрядов влево, (полный абсолютный
                ; адрес ячейки памяти 649 или 289h)
```

4.2.2. Косвенная адресация

Восемь вспомогательных регистров (*AR0–AR7*) (см. рис. 4.2) обеспечивают гибкую и удобную косвенную адресацию в процессорах *C5x*. В случае косвенной адресации адрес операнда находится в текущем вспомогательном регистре. Для выбора текущего вспомогательного регистра необходимо загрузить в регистр указателя вспомогательного регистра (*ARP*) значение в диапазоне от 0 до 7.

Над содержимым вспомогательных регистров можно производить ряд арифметических операций с помощью арифметического устройства вспомогательных регистров – **ARAU** *(Auxiliary Register Arithmetic Unit),* которое выполняет арифметические операции в фазе декодирования команды. Это позволяет сформировать адрес перед фазой декодирования следующей команды. Использование **ARAU** позволяет модифицировать содержимое *ARn* после использования его содержимого в текущей команде. Например, регистр

может быть инкрементирован или декрементирован. Возможны также другие виды модификации, рассматриваемые ниже.

С помощью косвенной адресации может быть адресована любая ячейка памяти в пределах 64К при помощи полного 16-разрядного значения, содержащегося во вспомогательном регистре. Адрес может быть загружен в регистр при использовании команды **LAR**. Вспомогательные регистры в *C5x* могут быть модифицированы командами **ADRK** (*add to auxiliary register short immediate*) или **SBRK** (*substract from auxiliary register short immediate*). Содержимое регистра *ARn* также может быть модифицировано инструкцией **MAR** (*modify auxiliary register*). Основным способом модификации является модификация, указываемая в поле косвенной адресации. Это возможно в любой команде, допускающей косвенную адресацию.

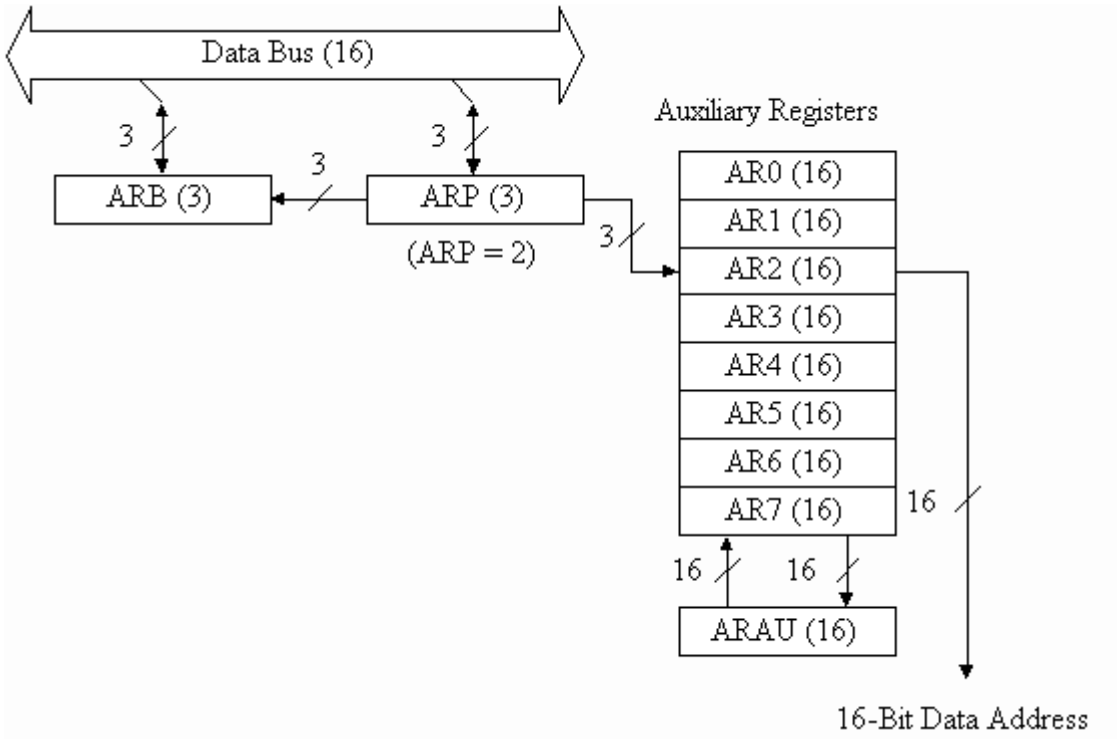

Рис. 4.2. Диаграмма косвенной адресации

Символы, используемые при косвенной адресации для обозначения видов модификации, включая бит-реверсивную адресацию, приведены в табл. 4.1. Сокращение *AR*(*ARP*) указывает на вспомогательный регистр, номер которого занесен в *ARP*.

Существует два основных вида модификации адреса при косвенной адресации:

- косвенная адресация с инкрементом и декрементом на 1;
- косвенная адресация с индексацией (изменением) на величину, содержащуюся в регистре *INDX.*

Во втором случае возможно в свою очередь два варианта:

- добавление или вычитание содержимого регистра *INDX;*
- добавление или вычитание содержимого регистра *INDX* по правилам бит реверсивной арифметики (обычно используется для реализации быстрых алгоритмов преобразования Фурье).

Во всех случаях содержимое вспомогательного регистра, указанного регистром *ARP*, используется как адрес операнда в памяти данных. *ARAU* осуществляет математические операции над содержимым указанного вспомогательного регистра.

Таблица 4.1

Символы, используемые при косвенной адресации

Символ	Значение
*	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти
*–	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти; после выполнения команды содержимое декрементируется (уменьшается) на 1
*+	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти; после выполнения команды содержимое инкрементируется (увеличивается) на 1
*0–	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти с последующим вычитанием из него содержимого регистра *INDX* после выполнения команды
*0+	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти с последующим сложением с содержимым регистра *INDX* после выполнения команды
*BR0–	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти с последующим вычитанием из него содержимого регистра *INDX* по правилам бит-реверсивной арифметики
*BR0+	Содержимое *AR*(*ARP*) используется как адрес ячейки памяти с последующим сложением с содержимым регистра *INDX* по правилам бит-реверсивной арифметики

После выполнения любой команды с косвенной адресацией в *ARP* может быть загружено новое значение. Если в *ARP* загружается новая величина, то старая величина загружается в буфер вспомогательного регистра состояния *ST1*.

Косвенная адресация с помощью вспомогательного регистра дает возможность изменения содержимого текущего вспомогательного регистра после выполнения команды. Изменением содержимого может быть увеличение или уменьшение его на 1 или на величину, задаваемую содержимым регистра *INDX*.

Для возможности реализации в процессоре C5x программ, написанных для процессоров семейства *C2x*, необходимо установить бит *NDX* в регистре *PMST* в 0. В архитектуре процессоров *C2x* текущий вспомогательный регистр мог быть инкрементирован или декрементирован значением, содержащимся в регистре *AR0*. Когда бит *NDX* сброшен в 0, любая модификация регистра *AR0* или загрузка его инструкцией **LAR** будет отражаться также и в регистрах *INDX* и *ARCR* (в эти регистры будет загружено то же самое значение, что и в регистр *AR0*). Последующие модификации текущего вспомогательного регистра будут производиться с использованием индексного значения регистра *INDX*. Таким образом достигается совместимость с существующим кодом для процессоров *C2x*. Бит *NDX* после сброса устанавливается в 0.

Бит-реверсивная адресация в *C5x* позволяет использовать алгоритмы вычисления БПФ. При этой адресации направление переноса в арифметическом устройстве вспомогательного регистра (*ARAU*) меняется. Если обычно перенос производится в старшие разряды числа, то в этом случае перенос идет в сторону младших разрядов. Содержимое регистра *INDX* добавляется или вычитается из текущего вспомогательного регистра. Обычное использование этого режима адресации при БПФ требует, чтобы сначала в регистр *INDX* было загружено значение, соответствующее половине размера массива данных и чтобы *AR*(*ARP*) был установлен в базовый адрес массива данных (1-й отсчет данных).

Косвенная адресация может быть использована во всех командах, кроме команд с непосредственными операндами или без операндов.

Следующие примеры иллюстрируют использование косвенной адресации:

Пример 1. ADD *+, 8

Сложение содержимого аккумулятора с содержимым ячейки памяти, на которую указывает текущий вспомогательный регистр, сдвинутым влево на 8 разрядов. Содержимое текущего вспомогательного регистра после выполнения команды увеличивается на 1.

Пример 2. ADD *, 8

То же, что и в примере 1, но модификация текущего вспомогательного регистра после выполнения команды не производится.

Пример 3. ADD *–, 8

То же, что и в примере 1, но содержимое текущего вспомогательного регистра после выполнения команды уменьшается на 1.

Пример 4. ADD *0+, 8

То же, что и в примере 1, но к содержимому текущего вспомогательного регистра после выполнения команды прибавляется содержимое регистра *INDX*.

Пример 5. **ADD *0–, 8**

То же, что и в примере 4, но из содержимого текущего вспомогательного регистра после выполнения команды отнимается содержимое регистра *INDX*.

Пример 6. **ADD *+, 8, AR3**

То же, что и примере 1, но в регистр *ARP* после выполнения команды загружается новое значение –3 (т. е. текущим вспомогательным регистром будет *AR3*).

Пример 7. **ADD *BR0–, 8**

Содержимое индексного регистра *INDX* после выполнения команды вычитается из содержимого текущего вспомогательного регистра по правилам бит-реверсивной арифметики.

Пример 8. ADD *BR0+, 8

Содержимое индексного регистра *INDX* после выполнения команды прибавляется к содержимому текущего вспомогательного регистра по правилам бит-реверсивной арифметики.

4.2.3. Непосредственная адресация

В случае непосредственной адресации операнд содержится непосредственно в слове команды. В процессорах *C5x* имеются однословные команды с непосредственной адресацией (8-битовые, 9-битовые и 13-битовые константы) и двухсловные (16-битные константы) длинные команды с непосредственной адресацией. В коротких командах непосредственный операнд находится прямо в слове. В длинных командах непосредственный операнд содержится во втором слове.

Команды процессора *C5x*, поддерживающие непосредственную адресацию, приведены в табл. 4.2.

Пример команды **RPT** с короткой непосредственной адресацией:

RPT #99 ; Повторение команды, следующей за *RPT,* 100 раз

В этом примере непосредственный операнд расположен непосредственно в командном слове.

Пример команды **RPT** с длинной непосредственной адресацией приведен ниже.

RPT #0FFFh ; Повторение команды, следующей за *RPT,* 1000h раз

В этом случае 16-битовая константа, задающая адрес, содержится в отдельном слове, следующем за словом команды.

Таблица 4.2

Инструкции процессора C50,

поддерживающие непосредственную адресацию

8-битовые константы	9-битовые константы	13-битовые константы	16-битовые константы
ADD ADRK LACL LAR RPT SBRK SUB	LDP	MPY	ADD AND APL CPL LACC LAR MPY
			OPL OR RPT RPTZ SPLK SUB XOR XPL

4.2.4. Циклическая адресация

Циклическая адресация представляет метод адресации, при котором происходит циклическое обращение к ячейкам некоторой области памяти (к ячейкам буфера), т.е. при достижении конца буфера автоматически происходит переход к его началу и наоборот [1]. В пределах буфера для указания адреса используется вспомогательный регистр (т.е. косвенный принцип адресации). При инкременте содержимого регистра (увеличении адреса) при достижении последней ячейки буфера в этот регистр автоматически заносится начальный адрес буфера.

Такая адресация используется при реализации многих алгоритмов, таких как свертка, корреляция, реализация линии задержки, КИХ-фильтрация. В этих алгоритмах циклический буфер используется для реализации скользящего окна, в котором содержатся данные для обработки.

На рис. 4.3 приведен алгоритм циклической адресации. Рисунок показывает, как будет распределяться последовательность отсчетов входного сигнала в буфере длиной 6 ячеек. В таком буфере, например, можно организовать линию задержки на 5T, где T – интервал дискретизации. Для этого надо последовательно считывать отсчеты x(0) и x(5), x(1) и x(6) и т.д.

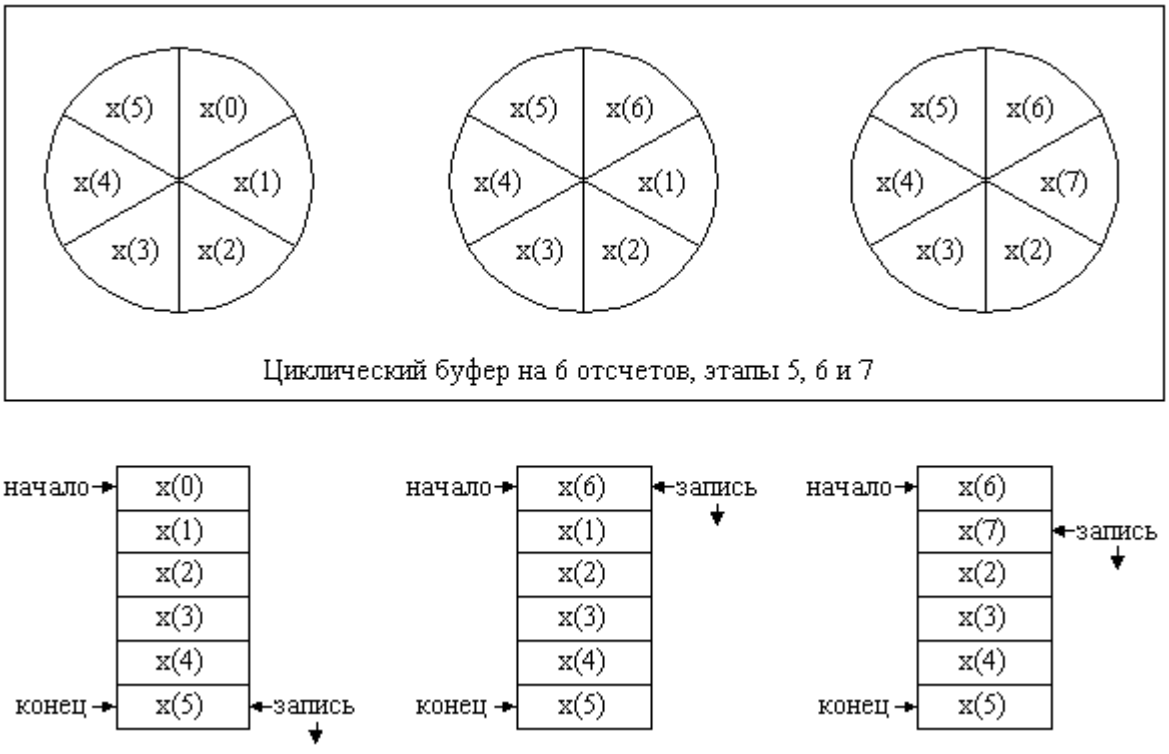

Рис. 4.3. Алгоритм циклической адресации

В процессорах семейства *C5x* имеется аппаратный механизм реализации циклических буферов, которые позволяют организовать два одновременно действующих циклических буфера, управляемых с помощью вспомогательных

регистров. Существует 5 регистров, с помощью которых осуществляется управление циклическими буферами:

- *CBSR1* – регистр начального адреса первого циклического буфера;
- *CBSR2* – регистр начального адреса второго циклического буфера;
- *CBER1* – регистр конечного адреса первого циклического буфера;
- *CBER2* – регистр конечного адреса второго циклического буфера;
- *CBCR* – регистр управления циклическими буферами.

Регистр управления (*CBCR*) определяет, какие вспомогательные регистры работают с данным буфером, разрешает/запрещает использование циклических буферов. Назначение отдельных битов этого регистра приведено в табл. 4.3.

Чтобы задать циклический буфер, необходимо вначале загрузить адреса начала и конца в соответствующие регистры циклического буфера, затем определить вспомогательный регистр, который будет работать указателем на ячейку памяти внутри циклического буфера. И в завершение установить бит разрешения работы соответствующего циклического буфера в регистре *CBCR*.

Таблица 4.3

Назначение битов регистра *CBCR*

Бит	Наименование	Функция
0-2	*CAR1*	Указывает, какой вспомогательный регистр работает с циклическим буфером 1
3	*CENB1*	Циклический буфер 1, разрешено = 1, запрещено = 0. При сбросе устанавливается в 0
4-6	*CAR2*	Указывает, какой вспомогательный регистр работает с циклическим буфером 2
7	*CENB2*	Циклический буфер 2, разрешено = 1, запрещено = 0. При сбросе устанавливается в 0

Примечание. Использование одного и того же вспомогательного регистра с обоими циклическими буферами может привести к непредсказуемым результатам.

Алгоритм изменения адреса для циклического буфера следующий (заметьте, что проверка содержимого вспомогательного регистра производится перед его модификацией):

Если (*ARn = CBER*) и (любая модификация *AR*),

то *ARn = CBSR*;

иначе *ARn* = *ARn* + шаг.

В добавление отметим :

- Если *ARn = CBER* и не производится модификация *AR*, текущий *AR* не модифицируется и остается равным *CBER*.
- Когда содержимое текущего вспомогательного регистра = CBER и производится любая модификация *AR* (инкремент или декремент с любым шагом), текущий *AR* будет установлен = *CBSR*.
- В случае использования модификации вспомогательного регистра с шагом, большим 1, измененный адрес, содержащийся во вспомогательном регистре, может выйти за пределы циклического буфера. Арифметическое устройство вспомогательных регистров *(ARAU)* не обнаруживает эту ситуацию и возврата на начало циклического буфера не происходит. Адрес после модификации должен попасть на конечную ячейку буфера.
- Изменение адреса циклического буфера при модификации текущего вспомогательного регистра может идти в любом направлении, но изменение должно идти в направлении от начала буфера (содержимого *CBSR*) к концу буфера (содержимое *CBER*). Соответственно для инкрементирования значения во вспомогательном регистре, значение в *CBER* должно быть больше, чем значение в *CBSR*, а для декрементирования значения во вспомогательном регистре, значение в *CBSR* должно быть больше, чем значение в *CBER*.
- Описанные выше ситуации показаны в примере 9.

Пример 9. Циклическая адресация:

```
                SPLK    #200h, CBSR     ; задание начального адреса циклического буфера1
                SPLK    #203h, CBER1    ; задание конечного адреса циклического буфера1
                SPLK    #0Eh, CBCR      ; указание на адресацию буфера с помощью
                                        ; регистра AR6
; Вариант 1
                LAR     AR6, #200h
                LACC    *               ; Загрузить в аккумулятор содержимое ячейки,
                                        ; указываемое AR6 = 200h (начало циклического
                                        ; буфера), содержимое AR6 не меняется
; Вариант 2
                LAR     AR6, #203h
                LACC    *               ;Загрузить в содержимое ячейки, указываемое
                                        ; AR6 = 203h (конец циклического буфера),
                                        ; содержимое AR6 не меняется
; Вариант 3
                LAR     AR6, #200h
                LACC    *+              ; Загрузить в аккумулятор содержимое ячейки,
                                        ; указываемое AR6 = 200h, затем, увеличить AR6
```

```
                                        ;на 1, после выполнения команды будет AR6 = 201h
; Вариант 4
                LAR     AR6, #203h
                LACC    *+              ; Загрузить в содержимое ячейки, указываемое
                                        ; AR6 = 203h, увеличить AR6 на 1, но так как
                                        ; AR6 = CBER1, то выполняется переход на начало
                                        ; буфера, будет AR6 = 200h
; Вариант 5
                LAR     AR6, #203
                LACC    *–              ; AR6 указывает на конец буфера, в результате
                                        ; попытки модификации в AR6 будет занесен
                                        ; не адрес 202h, а адрес начала буфера – AR6 = 200h
; Вариант 6
                LAR     AR6, #202h
                ADRK    #2              ; В результате модификации с шагом 2 адрес,
                                        ; содержащийся в AR6 выходит за пределы буфера,
                                        ; он станет равным  AR6 = 204h и будет далее
                                        ; меняться с шагом 2
; Вариант 7
                LAR     AR6, #203H
                ADRK    #2              ; Практически повторяется ситуация, описанная
                                        ; в варианте 6, однако, так как текущий адрес
                                        ; есть адрес конца буфера, то после команды
                                        ; будем иметь  AR6 = 200h – адрес начала буфера
```

4.3. Задание на самостоятельную подготовку

1. Подготовьте заготовку для отчета.

2. Изучите материалы теоретической справки и вышеприведенных примеров (п. 4.2).

3. Изучите текст программы lab4.asm, приведенной ниже.

```
                .version 50
                .mmregs
                .global direct1, direct2, direct3, direct4
                .global indirect1, indirect2, indirect3, indirect4
                .global circular1, circular2
                .sect   "Vectors"
RESET:          B       Init
ND1:            .sect   "offset"        ; Таблица смещений для регистра INDX
                .word   2
                .word   4
                .text
```

```
Init:       LDP     #0
            LACC    #00h
            OPL     #4,PMST         ; Устанавливаем бит NDX в 1,
                                    ; чтобы использовать регистр INDX
direct1:    LDP     #4              ; Страница памяти данных 4
            SACL    0000h           ; Прямая адресация
            ADD     #01h            ; Непосредственная адресация
            SACL    0000h
            ADD     #01h
            SACL    0002h
            ADD     #01h
            SACL    0003h
            ADD     #01h
direct2:    LDP     #5              ; Страница памяти данных 5
            SACL    0000h           ; Прямая адресация
            ADD     #01h            ; Непосредственная  адресация
            SACL    0000h
            ADD     #01h
            SACL    0002h
            ADD     #01h
            SACL    0003h
            ADD     #01h
direct3:    LDP     #4              ; Страница памяти данных 4
            SACL    0000h           ; Прямая адресация
            ADD     0000h           ; Прямая адресация
            SACL    0002h
            ADD     0002h
            SACL    0003h
            ADD     0003h
            SACL    0004h
            ADD     0004h
direct4:    LDP     #5              ; Страница памяти данных 5
            SACL    0000h           ; Прямая адресация
            ADD     0000h           ; Прямая адресация
            SACL    0001h
            ADD     0001h
            SACL    0002h
            ADD     0002h
            SACL    0003h
            ADD     0003h
indirect1:  MAR     *,AR0           ; Задание текущего вспомогательного регистра
            LAR     AR0,#200h       ; Установка начального значения AR0
            LACC    #00h            ; Прямая адресация
            SACL    *+              ; Косвенная адресация
            ADD     #01h
            SACL    *+
```

```
                ADD     #01h
                SACL    *+
                ADD     #01h
                SACL    *+
                ADD     #01h
indirect2:      MAR     *,AR1           ; Задание текущего вспомогательного регистра
                LAR     AR1,#280h       ; Установка начального значения AR1
                SACL    *+              ; Косвенная адресация
                ADD     #01h            ; Непосредственная адресация
                SACL    *+
                ADD     #01h
                SACL    *+
                ADD     #01h
                SACL    *–
indirect3:      LMMR    INDX,ND1        ; Запись значения в индексный регистр
                ADD     #03h            ; Непосредственная адресация
                NOP
                SACL    *0+             ; Косвенная адресация
                ADD     #03h
                SACL    *0+
                ADD     #03h
                SACL    *0+
                LMMR    INDX,ND1+1      ; Изменение значения в индексном регистре
                NOP
                ADD     #03h
                SACL    *0+
                ADD     #03h
                SACL    *0+
                ADD     #03h
                SACL    *0+
indirect4:      LACC    #00h            ; Непосредственная адресация
                LAR     AR2,#200h
                ADD     #04h
                SACL    *+,AR2          ; Косвенная адресация с изменением
                                        ; текущего вспомогательного регистра
                ADD     #04h
                SACL    *+,AR1
                ADD     #04h
                SACL    *+,AR2
                ADD     #04h
                SACL    *+,AR1
                ADD     #04h
                SACL    *+,AR2
                ADD     #04h
                SACL    *+,AR1
circular1:
```

```
Buff1       .bss    Buffer1,11          ; Резервирование места под первый буфер
Buff2       .bss    Buffer2,11          ; Резервирование места под второй  буфер
            LDP     #0                  ; DP = 0
            SPLK    #10h,BRCR           ; Количество повторений блока команд
            SPLK    #Buff1,cbsr1        ; Начальный адрес первого буфера
            SPLK    #(Buff2+10),cbsr2   ; Начальныйй адрес второго буфера
            SPLK    #(Buff1+10),cber1   ; Конечный адрес первого буфера
            SPLK    #Buff2,cber2        ; Конечный адрес второго буфера
            LAR     AR6,#Buff1          ; Инициализация вспомогательных регистров,
            LAR     AR3,#(Buff2+10)     ; которые будут работать с буферами
            MAR     *,AR6               ; Задание AR6 – текущим регистром
            SPLK    0BEh,cbcr           ; Разрешение работы циклических буферов
                                        ; и назначение AR6 для работы с первым буфером
                                        ; AR3 – для работы со вторым  буфером
            LACC    #1                  ; (ACC) = 1
            RPTB    end_block–1         ; Этот блок повторяется в цикле 16 раз
            SACL    *+,AR3              ; Сохранение содержимого аккумулятора
            SACL    *–,AR6              ; Сохранение содержимого аккумулятора
            ADD     #1                  ; Изменение содержимого аккумулятора
end_block:
circular2:  LDP     #0                  ; DP=0
            LACC    #1                  ; (ACC)=1
            SPLK    #15h,BRCR           ; Количество повторений блока команд
            SPLK    #Buff1,cbsr1        ; Стартовый адрес первого буфера
            SPLK    #(Buff2+10),cbsr2   ; Стартовый адрес второго буфера
            SPLK    #(Buff1+10),cber1   ; Конечный адрес первого буфера
            SPLK    #Buff2,cber2        ; Конечный адрес второго буфера
            LAR     AR6,#Buff1          ; Инициализация вспомогательных
            LAR     AR3,#(Buff2+10)     ; регистров, которые будут работать
                                        ; с буферами
            MAR     *,AR6               ; Назначение AR6 – текущим
            SPLK    0BEh,cbc            ; Разрешение работы циклических буферов и
                                        ; назначение AR6 для работы с первым буфером,
                                        ; AR3 – для работы со вторым буфером
            LMMR    INDX,ND1
            RPTB    end_block1–1        ; Этот блок повторяется в цикле 21 раз
            SACL    *0+,AR3             ; Сохранение содержимого аккумулятора
            SACL    *0–,AR6             ; Сохранение содержимого аккумулятора
            ADD     #1                  ; Изменение содержимого аккумулятора
            end_block1
            .end
```

4.4. Выполнение лабораторной работы

1. Скопируйте в свой рабочий директорий файлы:

- ***sim5x.exe*** – программа имитатора;
- ***alias.bat*** – файл с набором макрокоманд имитатора;
- ***siminit.cmd*** – управляющий файл для имитатора;
- ***lab4.asm*** – исходный текст программы лабораторной работы;
- ***lab4.cmd*** – командный файл компоновки.

2. Выполните трансляцию и компоновку программы ***lab4.asm***. Для трансляции наберите в командной строке команду

dspa.exe –lsc lab4.asm

Для компоновки наберите в командной строке команду

dsplnk.exe lab4.cmd

3. Запустить имитатор процессора, набрав ***sim5x.exe.*** При этом автоматически загрузится программа лабораторной работы ***lab4.out***, файл с макрокомандами ***alias.bat.***

4. Выполнить работу, следуя указаниям, приведенным ниже.

4.4.1. Указания к выполнению лабораторной работы

1. Выполните команду ***Go DIRECT1***. Пункты нижеследующего описания выполнения работы разделяются метками, одинаковыми с метками разделов программы, к которым пункты указаний относятся.

2. Во фрагменте ***DIRECT1*** используется прямая адресация ячеек памяти для записи результатов операций и непосредственная адресация слагаемого. Наберите команду ***DP*** – вывод указателя страницы памяти в окно WATCH. Наберите команду ***mem 0x200*** – вывод в окно ***MEMORY*** области памяти данных с адреса *0x200*.

Выполняйте программу по шагам до достижения очередной метки, наблюдая изменение содержимого ACC и памяти.

3. Во фрагменте ***DIRECT2*** при аналогичных командах изменяется страница памяти для размещения результатов: содержимое памяти по адресу *0x200* не меняется, изменилось значение ***DP***. Наберите
mem 0x280 для отражения страницы памяти 5 – область памяти с адреса *0x280*.

4. Во фрагменте ***DIRECT3*** используется прямая адресация памяти для результатов и прямая адресация слагаемых. Наберите команду
mem 0x200. Обратите внимание, что при прямой адресации слагаемых к содержимому *ACC* прибавляется не значение, стоящее в команде, а содержимое соответствующей ячейки памяти.

5. Во фрагменте ***DIRECT4*** по сравнению с предыдущим случаем произведена смена текущей страницы памяти данных. Наберите
mem 0x280 для просмотра содержимого этой страницы и продолжите выполнение программы.

6. Во фрагменте ***INDIRECT1*** используется косвенная адресация ячеек памяти для записи результатов операций и непосредственная адресация слагаемого. Наберите команду ***ARP*** – для отражения указателя текущего *ARn*. Наберите команду ***mem 0x200***. Следите за содержимым текущего *ARn* в окне *CPU* и изменением содержимого ячейки памяти, указываемой текущим вспомогательным регистром.

7. Во фрагменте ***INDIRECT2*** по сравнению с предыдущим случаем при аналогичных командах изменяется вспомогательный регистр, используемый для адресации и номера ячеек памяти для размещения результатов. Наберите *mem 0x280* для отображения содержимого ячеек.

8. Во фрагменте ***INDIRECT3*** используется модификация текущего *AR1* с использованием *RG INDX*. Наберите **NDX,** чтобы убедится, что режим совместимости с *C2x* отключен и используется независимое изменение содержимого регистров *AR0* и *INDX*. Обратите внимание: теперь содержимое текущего *RG* меняется с шагом, равным содержимому *INDX*, и запись информации в память идет в отличие от предыдущего случая сначала через одну ячейку, а затем и через три ячейки. Это определяется значениями, загружаемыми в *INDX* с помощью команд LMMR.

9. Во фрагменте ***INDIRECT4*** программы используется изменение текущего вспомогательного регистра в команде сохранения результата **SACL**. Запись содержимого *ACC* происходит попеременно в ячейки памяти, указываемые *AR1* и, *AR2*. Проверьте содержимое ячеек памяти с результатами вычислений. Для отображения на экране второй области памяти наберите команду ***mem1 0x200***.

10. Во фрагменте ***CIRCULAR1*** программы организуются два циклических буфера, которые адресуются с помощью вспомогательных регистров *AR6, AR3*. Эти буфера поочередно, в цикле, заполняются возрастающими значениями. Причем буфер 1 (регистр *AR6*) заполняется в сторону возрастающих адресов, а буфер 2 (регистр *AR3*) в сторону уменьшения адресов. Наберите команды **CBSR1, CBSR2, CBER1, CBER2** для отображения начальных и конечных адресов циклических буферов. Наберите команду ***mem 0x800*** для отображения в окне ***MEMORY*** области памяти, в которой находятся оба буфера. Выполните фрагмент программы в пошаговом режиме, проследите порядок изменения адресуемых ячеек памяти для буферов 1 и 2.

11. Во фрагменте ***CIRCULAR2*** повторяются команды предыдущего раздела. Отличие состоит в шаге изменения адресуемых ячеек памяти. В этом случае он определяется величиной, записанной в регистр *INDX*, – в данном случае 2.

Выполните фрагмент программы в пошаговом режиме, проследите порядок изменения адресуемых ячеек памяти для буферов 1 и 2.

12. По заданию преподавателя измените некоторые параметры программы:

- страницу памяти данных;
- непосредственно адресуемое значение;

- используемые для косвенной и циклической адресации вспомогательные регистры;

- длину циклических буферов;
- шаг изменения адреса при циклической адресации.

4.5. Содержание отчета

1. Текст программы с подробными комментариями.
2. Ответы на контрольные вопросы.

3. Изменения, внесенные в программу по указанию преподавателя, выводы по полученным при этом результатам.

Контрольные вопросы

1. Назовите основные методы адресации процессора *TMS320C5x*.
2. Поясните, как формируется адрес при прямой адресации.
3. Поясните, как формируется адрес при непосредственной адресации.
4. Поясните, как формируется адрес при косвенной адресации.
5. В чем особенность циклической адресации?
6. Как задать циклический буфер?
7. Как изменяются адреса при циклической адресации?

Приложение А

Цифровой процессор обработки сигналов TMS320C50

А.1. Главные особенности процессоров TMS320C5x

Различные процессоры семейства TMS320C5x имеют одинаковое ядро и отличаются составом и параметрами периферийных устройств. Процессоры семейства совместимы на уровне команд (ассемблера) с другими процессорами с фиксированной запятой – C1x, C2x и некоторыми другими. Основные особенности ЦПОС C5x перечислены ниже:

- длительность командного цикла 35/50 нс (время выполнения одиночной команды), 28,6/20 MIPS (миллионов команд в секунду);
- внутреннее ОЗУ типа SARAM (память с одиночным доступом): C50 – 9К × 16 бит, C51 – 1К × 16 бит, C53 – 3К × 16 бит;
- внутренняя память программ типа ROM: C50 – 2К × 16 бит, C51 – 8К × 16 бит, C53 – 16К × 16 бит;
- 1056 × 16 бит внутреннее ОЗУ двойного доступа типа DARAM;
- 224К – максимальное адресуемое пространство внешней памяти (64К программа, 64К данные, 64К ввод/вывод и 32К глобальные данные);
- 32-разрядное арифметико-логическое устройство (ALU), 32-разрядный аккумулятор (ACC), и 32-разрядный буфер аккумулятора (ACCB);
- 16-разрядное параллельное логическое устройство (PLU);
- 16 × 16 разрядный параллельный умножитесь с 32-разрядным регистром результата;
- восемь вспомогательных регистров со специализированным арифметическим модулем для косвенной адресации;
- 8-уровневый аппаратный стек;
- сдвигатели данных от 0 до 16 разрядов (вправо и влево) и инкрементное 64-разрядное сдвигающее устройство;
- возможность организации двух буферов с циклической адресацией;
- наличие команды повторения одиночной инструкции и команды повторения блоков программного кода;
- наличие операции пересылки содержимого блоков памяти для оптимального управления программой и данными;
- интервальный таймер;
- 64К параллельных портов ввода/вывода, 16 из которых отражены на память;
- 16 программируемых генераторов задержек для пространств программ, данных, ввода/вывода;
- наличие адресации с обратным распространением переноса (bit-reversed index-addressing mode) для выполнения БПФ.

А.2. Краткий обзор архитектуры процессора TMS320C50

TMS320C5x – высокопроизводительные цифровые процессоры обработки сигналов – базируются на модифицированной гарвардской архитектуре. В этой архитектуре используются два пространства памяти – для программ и для данных со своими шинами адреса и данных. По шине данных памяти программ из программной памяти пересылаются коды команд и непосредственные операнды. По шине данных памяти данных пересылаются данные от разнообразных модулей процессора (АЛУ, массива дополнительных адресных регистров и пр.) в память данных и наоборот. Имеются команды обмена данными между памятью программ и памятью данных.

В C5x поддерживается высокий уровень параллелизма. Так, пока данные обрабатываются в АЛУ, в арифметическом устройстве вспомогательных регистров может производиться инкремент или декремент содержимого этих регистров.

Процессор TMS320C5x выполняет арифметические команды, используя 32-разрядные АЛУ и аккумулятор. АЛУ – универсальный арифметический модуль, который оперирует 16-разрядными операндами (непосредственными или из памяти) или/и 32-разрядными из умножителя или аккумулятора. Аккумулятор используется для хранения результатов, поступающих из АЛУ, а также для ввода второго операнда в АЛУ. 32-разрядный аккумулятор разделен на две части (старшее слово – биты 31:16 и младшее – биты 15:0). Предусмотрены команды для сохранения старшего и младшего слов содержимого аккумулятора в памяти. Для быстрого временного сохранения содержимого аккумулятора имеется 32-разрядный буфер аккумулятора.

В дополнение к основному АЛУ имеется параллельный логический модуль (ПЛУ), который выполняет логические операции над данными, не оказывая влияния на содержимое аккумулятора. ПЛУ упрощает поразрядную установку, очистку и тестирование, используемые при управлении и при операциях над регистрами состояния (см. рис. А.1).

Аппаратный умножитель выполняет перемножение двух 16-разрядных слов с получением 32-разрядного результата за один командный цикл. Умножитель состоит из трех элементов: собственно умножителя, регистра результата (product register PREG) и временного регистра (TREG0). 16-разрядный TREG0 хранит множитель, 32-разрядный PREG содержит результат умножения. В зависимости от используемых команд значение множителя может быть загружено из памяти данных, памяти программ или непосредственно из команды. Аппаратный умножитель позволяет процессору эффективно выполнять основные операции обработки сигналов, такие как свертки, корреляции, фильтрацию.

Масштабирующий сдвигатель имеет 16-разрядный вход, соединенный с шиной данных, и 32-разрядный выход, соединенный со входом АЛУ. Масштабирующий сдвигатель производит левый сдвиг входных данных от 0 до 16 бит. Сдвиг может задаваться в команде или с помощью указателя

динамического сдвига (регистра TREG1). Младшие разряды при сдвиге заполняются нулями, старшие – или нулями (SXM=0), или расширением знака, в зависимости от состояния бита SXM (sign-extension mode bit) в регистре состояния ST1. Дополнительные возможности сдвига позволяют процессору выполнять численное масштабирование, расширение разрядности и операции по предотвращению переполнения.

8-уровневый аппаратный стек служит для хранения содержимого программного счетчика при выполнении процедур и подпрограмм обработки прерываний. При прерываниях регистры (ACC, ARCR, INDX, PMST, PREG, ST0, ST1, TREGs) помещаются в одноуровневый стек и извлекаются из него при возврате из прерывания, что предоставляет возможность переключать контекст без потери времени.

Структурная схема процессора TMS320C50 представлена на рис. А.1. На структурной схеме показаны принципиальные блоки и пути данных в семействе процессоров C5x.

Перечень внутренних аппаратных средств процессора и обозначений элементов схемы приводится в табл. А.1.

А.2.1. Ядро CPU

Процессоры семейства C5x поддерживают совместимость исходного текста программы на языке ассемблера с текстами программ для первых поколений процессоров C1x и C2x, а также совместимость аппаратных решений. Усовершенствования включают 32-разрядный буфер аккумулятора, дополнительные возможности масштабирования и новые команды для эксплуатации дополнительных аппаратных средств. К усовершенствованиям также относятся независимый параллельный логический модуль (PLU) для выполнения логических операций, набор регистров переключения контекста для обеспечения нулевого времени ожидания при подключении подпрограммы обработки прерывания (ISRs). Управление данными было улучшено с помощью нового блока команд, предназначенных для работы с регистрами, расположенными в памяти (C5x имеет 28 регистров основного центрального процессора и 16 портов ввода/вывода, которые адресуются как ячейки памяти).

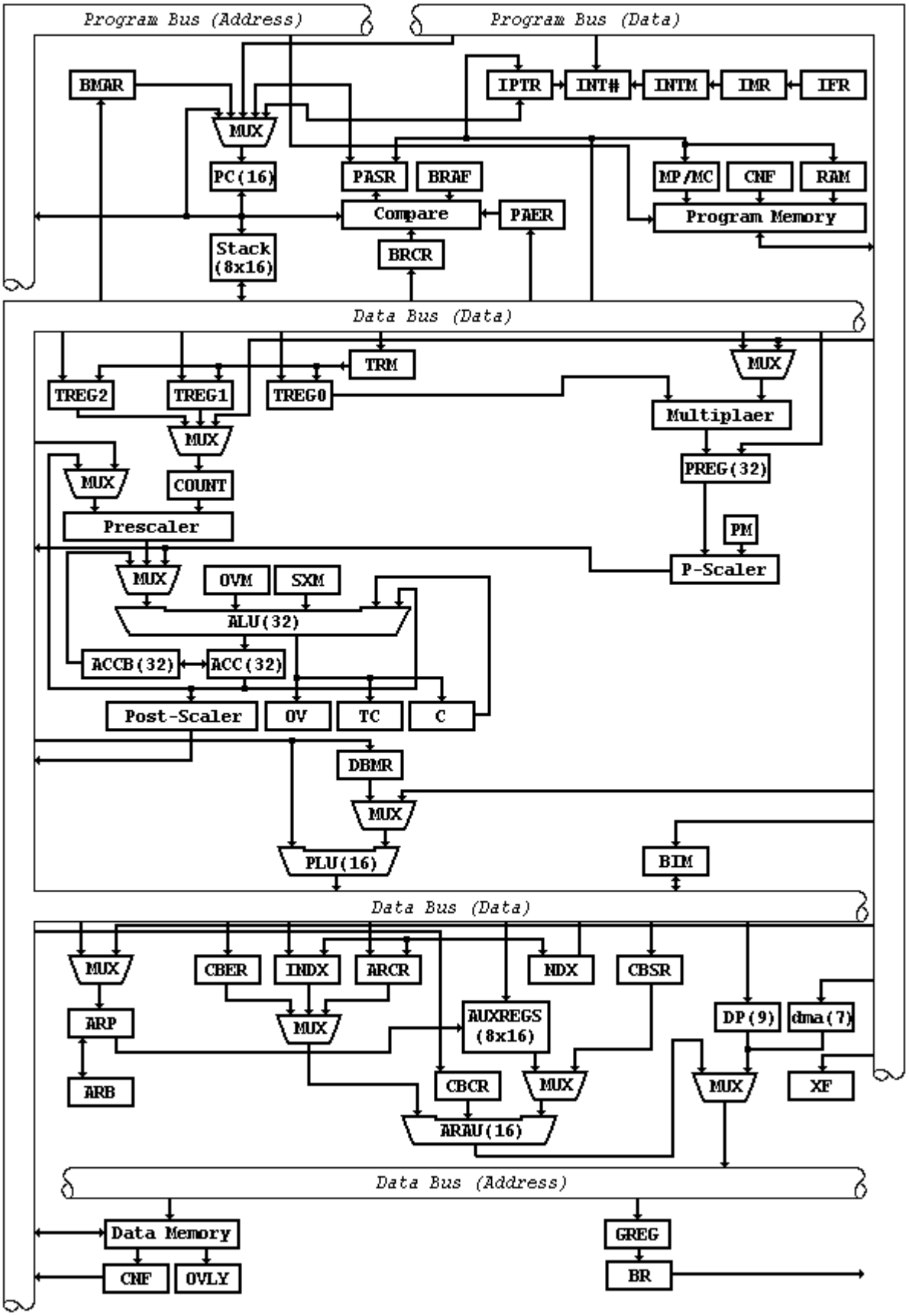

Рис. А.1. Структурная схема

Таблица А.1

Перечень внутренних аппаратных средств

Модуль	Символ	Функциональное назначение
Аккумулятор	ACC(32) ACCH(16) ACCL(16)	32-разрядный аккумулятор разделен на две части по 16 разрядов: старшую ACCH(16) и младшую ACCL(16). Используется для сохранения результатов операций с АЛУ
Буфер аккумулятора	ACCB(32)	Регистр предназначен для временного хранения 32-разрядного содержимого аккумулятора. Регистр связан с АЛУ и может участвовать в арифметических и логических операциях
Арифметико-логическое устройство	ALU	32-разрядное АЛУ для выполнения арифметических и логических операций
Арифметическое устройство вспомогательных регистров	ARAU	Беззнаковое 16-разрядное арифметическое устройство предназначено для вычисления адресов при косвенной адресации, работает с индексными, адресными регистрами и регистрами сравнения
Дополнительный регистр сравнения	ARCR(16)	16-разрядный регистр предназначен для сравнения косвенных адресов
Файл вспомогательных адресных регистров	AUXREGS	Регистровый файл содержит восемь 16-разрядных адресных регистров, используемых для проведения косвенной адресации, временного хранения данных или целочисленной арифметики с использованием ARAU
Буфер вспомогательного адресного регистра	ARB(3)	3-разрядный регистр захватывает предыдущее значение ARP. Входит в регистр ST1

Указатель текущего вспомогательного регистра	ARP(3)	3-разрядный регистр используется как указатель на активный вспомогательный адресный регистр. Входит в регистр управления ST0
Адресный регистр блоковых пересылок	BMAR(16)	16-разрядный регистр содержит адрес при выполнении блоковых пересылок и операций умножения/суммирования
Флаг активности повторения блока	BRAF(1)	Индицирует активность режима повторения блока программного кода. Бит устанавливается, когда выполняется инструкция RPTB, и сбрасывается, когда регистр BRCR декрементируется до нуля. Бит содержится в регистре PMST
Регистр адреса окончания блока повторения	PAER(16)	16-разрядный расположенный в памяти регистр содержит адрес окончания сегмента кода, подлежащего выполнению в команде повторения участка программы
Регистр начального адреса блока повторения	PASR(16)	16-разрядный расположенный в памяти регистр содержит начальный адрес сегмента кода, подлежащего выполнению в команде повторения участка программы

Продолжение табл. А.1

Модуль	Символ	Функциональное назначение
Счетчик числа повторений участка программного кода	BRCR(16)	16-разрядный расположенный в памяти регистр, используется для задания числа повторений участка программного кода
Интерфейсный	BIM	Буферный интерфейс используется для

модуль шины		передачи данных по внутренним шинам программ и данных
Сигнал запроса шины	BR	Сигнал индицирует, что доступ осуществляется к глобальным данным, определенным через регистр GREG
Перенос	C	Бит хранит значение сигнала переноса АЛУ. Располагается в регистре ST1
Центральное арифметико-логическое устройство	CALU	Группа устройств, в которую входят: АЛУ, умножитель, аккумулятор и сдвигатели (описаны в данной таблице)
Управляющий регистр циклиских буферов	CBCR(8)	8-разрядный регистр используется для разрешения/запрещения циклических буферов и для определения вспомогательных адресных регистров, используемых в циклической адресации
Начальные адреса циклических буферов	CBSR(16)	Два 16-разрядных регистра (CBSR1, CBSR2) используются для задания начальных адресов буферов
Конечные адреса циклических буферов	CBER(16)	Два 16-разрядных регистра (CBER1, CBER2) используются для задания конечных адресов буферов
Компаратор с программным адресом	COMPARE	Производит циклическое сравнение содержимого PC с содержимым PAER, если активен флаг BRAF. Если равенство выполняется, то в PC загружается содержимое PASR
Конфигурация RAM	CNF	Бит устанавливает конфигурацию внутренней DARAM в программное пространство или пространство данных
Шина данных	DATA	16-разрядная шина используется для пересылок данных
Память данных	DATA MEMORY	Блок обозначает память данных, используемую процессором. Ссылка идет к внутренней и внешней памяти
Шина адреса памяти данных	DATA ADDRESS	Используется для передачи адреса при доступе к памяти данных

Регистр непосредственного адреса в памяти данных	dma(7)	7-разрядный регистр указывает непосредственный адрес в пределах страницы из 128 слов
Указатель на страницу в памяти данных	DP(9)	9-разрядный регистр содержит адрес страницы в 128 слов (всего 512 страниц)
Бит конфигурации RAM	RAM(1)	Установкой этого бита SARAM конфигурируется в пространство данных

Продолжение табл. А.1

Модуль	Символ	Функциональное назначение
Шина прямого адреса памяти данных	DRB(16)	16-разрядная шина используется для передачи прямого адреса, полученного путем конкатенации DP и DMA
Регистр битовых манипуляций	DBMR(16)	16-разрядный расположенный в памяти регистр, используется для маскирования выходов PLU при отсутствии длинного непосредственного значения
Динамический указатель бита	TREG2(4)	4-разрядный расположенный в памяти регистр хранит динамический указатель бита для команды BITT
Динамический счетчик сдвига	TREG1(5)	5-разрядный регистр хранит динамический счетчик сдвига, используемого PRE-SCALER для сдвига данных поступающих в АЛУ
Внешний флаг	XF(1)	Бит показывает уровень установленный на внешнем контакте и располагается в ST1
Регистр распределения глобальной памяти	GREG(8)	8-разрядный расположенный в памяти регистр используется для спецификации размера пространства глобальной памяти
Бит режима захвата (Hold mode)	HM(1)	Бит располагается в регистре ST1 и определяет, будет ли остановлен CALU, когда сигнал HOLD инициирует режим ПДП
Индексный регистр	INDX(16)	16-разрядный расположенный в памяти регистр определяет размер приращения при модификации косвенного адреса. При адресации с обратным распространением переноса регистр определяет размер массива

Флаг разрешния индексного регистра	NDX(1)	Бит определяет, будет ли запись/модификация регистра AR0 производить также запись/модификацию регистров INDX и ARCR для обеспечения совместимости с C25. Бит расположен в PMST
Регистр флажков прерываний	IFR(16)	16-разрядный расположенный в памяти регистр используется для запоминания активных сигналов прерываний
Бит маски прерываний	INTM(1)	Бит полностью маскирует или разрешает все прерывания. Располагается в регистре ST0
Номер прерывания	INT#(4)	Номер прерывания используется в CPU при активизации прерывания. Значение номера может приходить от аппаратных схем или задаваться в программе в инструкции INTR
Указатель прерываний	IPTR(5)	Пять бит задают конкретную страницу, в которой располагается таблица векторов прерываний. Эти биты располагаются в регистре PMST
Регистр маски прерываний	IMR(16)	16-разрядный расположенный в памяти регистр используется для маскирования прерываний
Микростек	MCS(15-0)	Однословный стек используется для временного хранения содержимого PFC, пока PFC используется для операций блоковых передач, умножения/суммирования и чтения/записи таблиц

Продолжение табл. А.1

Модуль	Символ	Функциональное назначение
Бит режима микропроцессора /микрокомпьютера	MP/MC	Бит располагается в регистре PMST и показывает включение внутреннего ПЗУ в программное адресное пространство
Мультиплексоры	MUX	Используются для выбора источника операнда для шин и различных устройств
Умножитель	MULTIPLIER	16× 16 разрядный параллельный умножитель
Флаг	OV(1)	Бит располагается в ST0 и показывает переполнение при арифметических

переполнения		операциях в АЛУ
Бит режима переполнения	OVM(1)	Бит располагается в ST0 и определяет режим переполнения (с насыщением или без насыщения)
Перекрытие в пространстве данных	OVLY(1)	Бит располагается в PMST и определяет, будет ли внутренняя SARAM располагаться в пространстве данных
Параллельное логическое устройство	PLU	16-разрядное логическое устройство, которое выполняет логические операции с длинным непосредственным значением или содержимым регистра DMBR параллельно с работой CALU и не оказывает влияния на содержимое CALU
Счетчик предварительной выборки	PFC (15-0)	16-разрядный счетчик используется для предварительной выборки команд программы. PFC содержит адрес команды, которая должна быть выбрана из памяти. Также PFC может адресовать программную память при командах пересылки блоков, умножения/суммирования
Регистр-счетчик предварительного масштабирования	COUNT(4)	4-разрядный регистр содержит значение для операций предварительного масштабирования. Регистр загружается из динамического счетчика сдвига или из команды. При применении команд битового сканирования регистр загружается из динамического указателя разряда
Регистр результата	PREG(32)	32-разрядный регистр результата используется для сохранения результата умножения. Доступ к старшей и младшей части PREG может осуществляться раздельно
Программная шина	PROG DATA	16-разрядная шина используется для пересылки команд (и данных для команд умножения/суммирования)
Программный счетчик	PC(16)	16-разрядный программный счетчик используется для адресации программной памяти
Программная память	PROGRAM MEMORY	Блок объединяет всю доступную программную память
Адресная шина программной	PROG ADDRESS	16-разрядная шина используется для пересылки адреса при доступе в

памяти		программную память

Окончание табл. А.1

Модуль	Символ	Функциональное назначение
Сдвигатель предвательного масштабирования	PRE-SCALER	Сдвигатель используется для левого сдвига от 0 до 16 бит при предварительном масштабировании данных, поступающих в АЛУ, а также при выравнивании данных и при сдвиге вправо от 0 до 16 бит содержимого ACC
Сдвигатель масштабирования результата	POST-SCALER	Сдвигатель используется для левого сдвига (от 0 до 7 бит) результата, поступающего из CALU
Сдвигатель результата	P-SCALER	Сдвигатель используется для сдвига влево на 0,1 или 4 бита при необходимости в отбрасывании лишних знаковых разрядов, образующихся при операции умножения, а также для сдвига вправо на 6 разрядов для предотвращения переполнения
Режим сдвига результата	PM(2)	Два бита определяют режим сдвигателя результата, располагаются в регистре ST1
Счетчик повторений	RPTC(16)	16-разрядный счетчик используется для управления циклическим исполнением одиночной команды
Режим расширения знака	SXM(1)	Бит располагается в ST1 и задает использование режима расширения знака при арифметических операциях
Стек	STACK	8-уровневый 16-разрядный аппаратный стек используется для сохранения PC при выполнении подпрограмм, может быть использован для сохранения данных
Статусные регистры	ST0, ST1, PMST	Три 16-разрядных статусных регистра содержат биты статуса и управления
Регистр временного множителя	TREG0(16)	16-разрядный регистр используется для временного хранения операнда для умножителя

Бит разрешения временных регистров	TRM(1)	Бит определяет, будут ли по команде LT загружаться все три регистра TREGs(0,1,2) для обеспечения совместимости с C25 или только регистр TREG0
Бит теста/управления	TC(1)	Бит располагается в ST1 и хранит результаты операций тестирования битов

A.2.2. Внутреннее ПЗУ

Процессор C50 имеет встроенное ПЗУ емкостью 2К × 16. Эта память содержит загрузчик для перекачки программ из более медленного внешнего ПЗУ в более быстрое внутреннее или внешнее ОЗУ. Загрузчик работает, если при сбросе на контакт MP/MC подан сигнал низкого уровня. После загрузки программы в оперативную память, ПЗУ начальной загрузки может быть удалено из программного пространства памяти через бит MP/MC в регистре состояния PMST. Если при сбросе ПЗУ не выбрано, C50 начинает выборку команд из памяти вне кристалла.

Процессор C51 имеет внутренний ROM емкостью 8К × 16, C53 – 16К × 16. Эта память может использоваться для хранения определенных программ.

A.2.3. Внутренняя оперативная память данных

Процессоры семейства C5x имеют ОЗУ двойного доступа емкостью 1056x16. К этой оперативной памяти можно обращаться дважды за один машинный цикл. Этот блок памяти предназначен, прежде всего, для хранения данных, но, когда есть необходимость, может использоваться для хранения программы. Возможны два способа конфигурации этого блока:

- Все 1056× 16 бит как память данных.
- 544× 16 бит – память данных; 512× 16 – память программ.

Конфигурация выбирается через установку бита CNF в регистре состояния ST1.

A.2.4. Внутреннее ОЗУ программ/данных

Процессор C50 имеет внутреннее ОЗУ емкостью 9К× 16, C51 –1К× 16. Эта память конфигурируется в пространство памяти данных или программ. Код может выполняться с максимальным быстродействием, только если он загружен в эту память.

A.2.5. Защита памяти на кристалле

Поколение процессоров C5x имеет опцию защиты содержания блоков памяти на кристалле. Когда защита установлена, команды из внешней памяти не могут обращаться к пространствам памяти на кристалле.

A.2.6. Программируемый генератор задержек

Программируемая логика ожидания предназначена для обеспечения подключения к процессорам C5x более медленной памяти и устройств ввода/вывода. Эта схема состоит из 16 программируемых пользователем генераторов задержек. Они могут быть настроены на генерацию 0, 1, 2, 3 или 7 циклов ожидания при работе с медленной внешней памятью.

A.2.7. Параллельные порты ввода/вывода

Каждое устройство семейства C5x имеет 64K 16-разрядных портов ввода/вывода, 16 из которых адресуются как ячейки памяти. К любому порту доступ осуществляется по командам IN и OUT. К портам ввода/вывода, расположенным в памяти, обращаться можно также с любой командой, которая читает или записывает в память данных. Сигнал IS используется для индикации команд чтения/записи в пространство ввода/вывода. При минимальных затратах на схемы декодирования адреса процессоры семейства C5x могут взаимодействовать с внешними устройствами через порты ввода/вывода.

A.2.8. Последовательные порты ввода-вывода

Устройства семейства C5x имеют два встроенных быстродействующих последовательных порта ввода/вывода. Эти порты способны выполнять операции с частотой до 1/4 частоты машинного такта (CLKOUT1). Одна из двух схем – синхронный, дуплексный последовательный порт. Приемник и передатчик имеют двойную буферизацию и индивидуально управляются маскируемыми внешними сигналами прерывания. Кадры передачи/приема могут иметь длину – байт или слово. Вторая схема – дуплексный последовательный порт, который может быть сконфигурирован для синхронной передачи/приема (аналогично первой схеме) или для выполнения операций коллективного доступа с разделением времени (TDM port). Последовательный TDM порт обычно используется в прикладных программах многопроцессорной системы.

A.2.9. Аппаратный таймер

Устройства семейства C5x имеют встроенный 16-разрядный таймер с дополнительным 4-разрядным регистром деления частоты. Таймер работает от частоты CLKOUT1 и используется для генерации периодических прерываний CPU. Таймер работает как вычитающий счетчик. Его можно также использовать для генерации внешних функций через контакт TOUT.

A.2.10. Маскируемые прерывания

Устройства семейства C5x имеют четыре линии внешних прерываний, а также пять внутренних прерываний: прерывание от таймера и четыре прерывания от последовательных портов ввода/вывода.

A.3. Организация памяти процессоров TMS320C5x

Общее адресное пространство в процессорах семейства C5x составляет 224К 16-разрядных слов. Общее пространство делится на четыре сегмента:

- 64К – память программ;
- 64К – память локальных данных;
- 32К – память глобальных данных;
- 64К – порты ввода/вывода.

Принцип параллельной обработки, заложенный в архитектуру C5x, позволяет выполнять до трех операций с памятью за один машинный цикл:

- выборку команды;
- выборку операнда;
- запись операнда.

A.3.1. Пространство памяти

Процессоры семейства C5x базируются на модифицированной гарвардской архитектуре.

Память C5x организована в четыре индивидуально выбираемых пространства: программа, локальные данные, глобальные данные, порты ввода/вывода. Эти подпространства образуют общее адресное пространство в 224К слов.

Процессоры семейства C5x имеют внутреннюю память (память внутри кристалла) для повышения быстродействия и интеграции. Кроме того, C50 содержит 2К слов загрузочного ПЗУ, 9К слов ОЗУ одинарного доступа для программ/данных (SARAM) и 1056 слов ОЗУ двойного доступа (DARAM). Загрузочное ПЗУ располагается в адресном пространстве по адресу 0 и включает тест процессора и загрузочный код. ОЗУ одиночного доступа может быть отражено в адресное пространство программ и/или данных и располагается по адресу 0800h. SARAM требует одного машинного цикла для каждой операции чтения/записи. В ОЗУ двойного доступа запись и чтение могут проводиться в одном цикле. 1056 слов DARAM сконфигурированы в 3 блока:

- блок 0 (B0) – 512 слов располагается по адресам 0100h–02FFh в пространстве локальных данных и по адресам 0FE00h–0FFFFh в программном пространстве;
- блок 1 (B1) – 512 слов – располагается по адресам 0300h–04FFh в пространстве локальных данных;

- блок 2 (B2) – 32 слова – располагается с адреса 0060h в пространстве локальных данных.

Карта памяти для процессора C50 показана на рис. А.2.

А.3.2. Конфигурирование памяти

Пространство памяти может быть переконфигурировано с помощью установки или сброса битов конфигурации RAM, MP/MC, OVLY в регистре PMST и бита CNF в регистре ST1.

Возможные варианты конфигурации памяти программ и данных показаны в табл. А.2 и А.3, соответственно.

А.3.3. Регистры, расположенные в памяти

64К слов локальной памяти данных включают расположенные в памяти регистры устройства. Эти регистры размещаются на 0-й странице памяти данных. Страница 0 имеет пять секций для регистровых банков:

- регистры ядра CPU (28 регистров), доступ к ним возможен без циклов ожидания и без передачи по шине данных;
- периферийные регистры – регистры данных и управления встроенными периферийными устройствами;
- зарезервированная область под регистры, используемые при тестировании и эмуляции;
- 16 параллельных портов ввода / вывода;
- область в 32 слова (блок B2 DARAM) для хранения временных переменных.

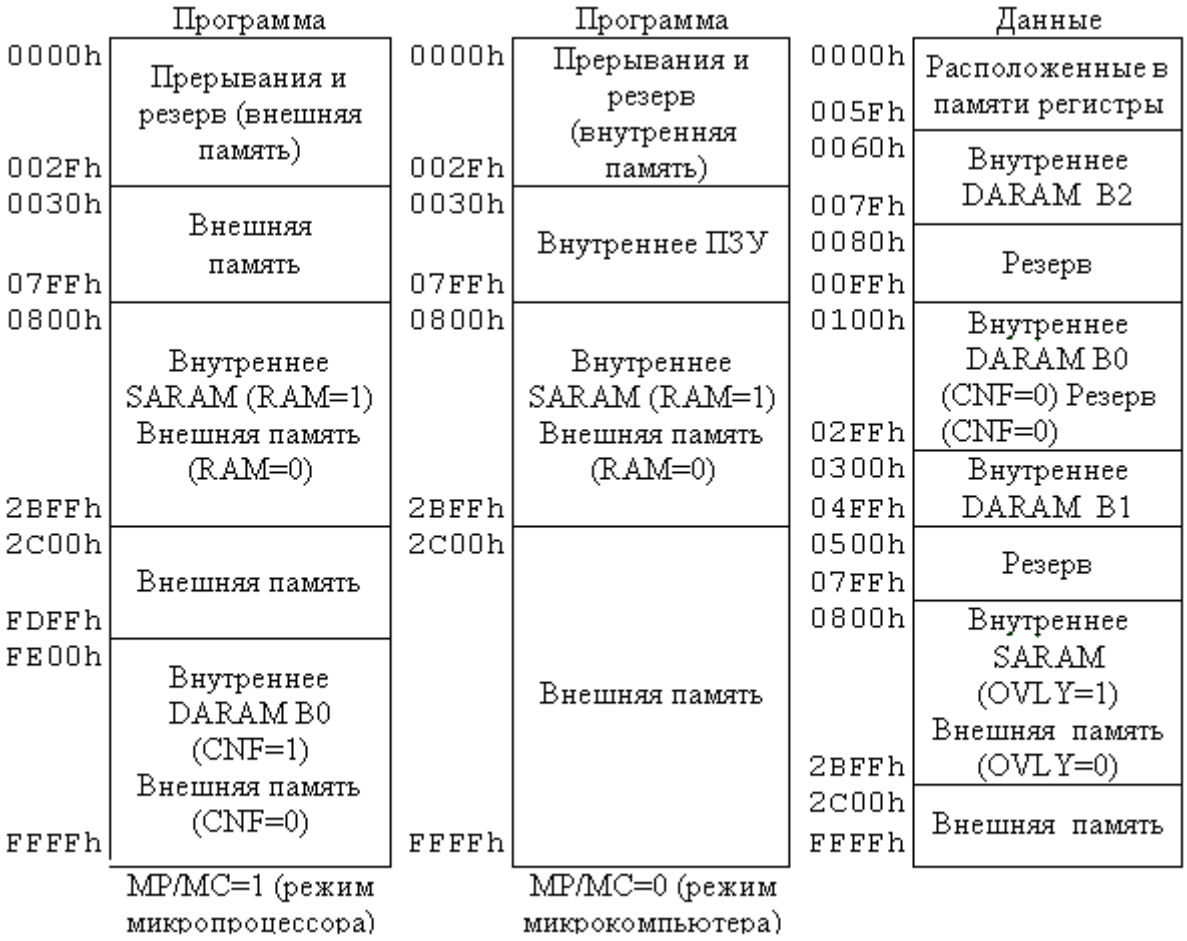

Рис. А.2. Карта памяти ЦПОС TMS320C50.

Таблица А.2

Управление конфигурацией памяти программ

CNF	RAM	MP/MC	ROM	SARAM	DARAM B0	Внешняя память
0	0	0	0000–07FF			0800–FFFF
0	0	1				0000–FFFF
0	1	0	0000–07FF	0800–2BFF		2C00–FFFF
0	1	1		0800–2BFF		0000–07FF 2C00–FFFF
1	0	0	0000–07FF		FE00–FFFF	0800–FDFF
1	0	1			FE00–FFFF	0000–FDFF

1	1	0	0000–07FF	0800–2BFF	FE00–FFFF	2C00–FDFF
1	1	1		0800–2BFF	FE00–FFFF	0000–07FF 2C00–FDFF

Таблица А.3

Управление конфигурацией памяти данных

CNF	OVLY	DARAM B0	DARAM B1	DARAM B2	SARAM	Внешняя память
0	0	100h–2FFh	300h–4FFh	60h–7Fh		800h–FFFFh
0	1	100h–2FFh	300h–4FFh	60h–7Fh	800h–2BFFh	2C00h–FFFFh
1	0	–	300h–4FFh	60h–7Fh		800h–FFFFh
1	1	–	300h–4FFh	60h–7Fh	800h–2BFFh	2C00h–FFFFh

А.3.4. Режимы адресации памяти

В процессорах семейства С5х используются следующие основные режимы адресации памяти:

- прямая адресация;
- косвенная адресация;
- адресация регистров, расположенных в памяти;
- непосредственная адресация.

При прямой адресации в семи младших разрядах командного слова указывается адрес ячейки данных. При этом адрес может быть указан только в пределах текущей страницы, содержащей 128 ячеек памяти. Номер текущей страницы содержится в указателе страницы (регистр DP), который загружается программно. При выполнении команды старшие 9 разрядов адреса считываются из регистра DP и к ним пристыковывается прямой адрес, указанный в команде.

Косвенная адресация памяти осуществляется в общем случае через восемь 16-разрядных вспомогательных адресных регистров (AR0–AR7), загрузка и модификация которых в общем случае осуществляется программно. Выбор текущего адресного регистра осуществляется при помощи 3-разрядного

указателя регистра (ARP). Модификация содержимого вспомогательных адресных регистров производится с использованием дополнительного арифметического устройства (ARAU). Модификация регистра происходит после использования содержимого регистра в команде в том же цикле, в котором выполняется команда. Кроме того, при выполнении команды указатель ARP может быть загружен новым значением.

В командах с косвенной адресацией используются следующие символы:

***** – содержимое текущего регистра используется как адрес памяти данных;

***–** – содержимое текущего регистра используется как адрес памяти данных, а затем декрементируется;

***+** – содержимое текущего регистра используется как адрес памяти данных, а затем инкрементируется;

***0–** – содержимое текущего регистра используется как адрес памяти данных, а затем из него вычитается содержимое индексного регистра INDX или при режиме, совместимом с C25, содержимое регистра AR0;

***0+** – содержимое текущего регистра используется как адрес памяти данных, а затем к нему прибавляется содержимое индексного регистра INDX или при режиме, совместимом с C25, содержимое регистра AR0;

***BR0–** – содержимое текущего регистра используется как адрес памяти данных, а затем из него вычитается содержимое индексного регистра INDX или при режиме, совместимом с C25, содержимое регистра AR0 (с обратным распространением переноса от старших разрядов к младшим);

***BR0+** – содержимое текущего регистра используется как адрес памяти данных, а затем к нему прибавляется содержимое индексного регистра INDX или при режиме, совместимом с C25, содержимое регистра AR0 (с обратным распространением переноса от старших разрядов к младшим).

Применение режимов адресации *BR0–, *BR0+ существенно ускоряет реализацию алгоритмов БПФ.

В режиме непосредственной адресации операнд находится в командном слове (при работе с короткими константами) или в слове, следующем за командным (при работе с длинными константами). Длина коротких констант зависит от кода операции и может быть 3, 8, 9, 13 бит. Длинная константа всегда имеет размер 16 бит.

Процессоры C5x обладают несколькими режимами адресации с определенными особенностями.

Режим адресации расположенных в памяти регистров отличается от изложенной выше прямой адресации тем, что обращение ведется всегда к нулевой странице данных, независимо от содержимого регистра DP. (Пример: **LAMM 07h** ; ACC = PMST).

В режиме регистровой адресации блоков памяти, используемом при операциях пересылки блоков данных, адрес одного из операндов может формироваться обычным образом, а в качестве второго может использоваться содержимое регистра BMAR. В случае, если не используется этот регистр при пересылках блоков данных, второй адрес берется из непосредственного операнда, следующего за командным словом.

Устройства C5x поддерживают также операции работы с двумя циклическими буферами (циклическую адресацию), в этом случае дополнительный адресный регистр будет перезагружаться начальным значением из регистра CBSR1/2 при превышении значения, указанного в регистре CBER1/2. Для управления работой с циклическими буферами используется регистр управления CBCR.

Более подробно режимы адресации рассмотрены в лабораторной работе 4.

A.4. Системное управление

Системное управление в C5x осуществляется через программный счетчик, аппаратный стек, внешний сигнал сброса, прерывания, регистры состояния и счетчики повторений.

A.4.1. Генерация программного адреса

Процессоры C5x имеют 16-разрядный программный счетчик (PC) и 8-уровневый аппаратный стек для хранения PC. Программный счетчик адресует внешнюю и внутреннюю программную память при выборке команд. Стек используется при исполнении прерываний и подпрограмм.

Регистр PC адресует программную память через программную шину чтения/записи PAB. По этой шине инструкции загружаются в регистр команд IREG. Когда IREG загружен, PC готов к выборке следующей команды.

Программный счетчик может быть загружен несколькими путями:

- при последовательном исполнении кода программы счетчик инкрементируется;

- при выполнении команд перехода счетчик загружается из длинного непосредственного операнда, следующего за инструкцией перехода;
- при выполнении подключения процедуры PC+2 помещается в стек, а программный счетчик загружается длинным непосредственным значением, следующим за командой CALL;

- при выполнении команды возврата адрес выталкивается из стека;
- при выполнении команд BACC, CALA программный счетчик загружается из аккумулятора;
- при выполнении прерываний PC загружается аналогично командам подключения процедур.

По шине PAB могут также пересылаться данные, хранимые в пространстве программ или данных. Это возможно при повторяющихся инструкциях для выборки второго операнда параллельно с шиной данных. Когда идет повторение выполнения команды, массив данных адресуется через PAB. При этом последовательный доступ к ячейкам осуществляется инкрементированием PC.

Операции пересылки блоков также используют шину программ и шину данных. Есть возможность выборки последующего операнда во время записи предыдущего. При использовании этих команд загрузка PC производится из регистра BMAR или из длинного непосредственного операнда. При чтении/записи таблиц загрузка PC осуществляется из аккумулятора.

Для реализации условных переходов C5x имеют полный набор условий подключения процедур, выполнения команд условных переходов и команд возврата. Множественные условия выполняются по логическому “И”.

Для выполнения команд условного перехода требуется до четырех машинных циклов. При этом в системе команд есть команды так называемых задержанных переходов (в этом случае следующая за командой перехода инструкция безусловно выполняется во время обработки команды перехода). Также существует команда пропуска инструкции. Ее время выполнения – 1 машинный цикл. При выполнении условия следующая за ней команда пропускается. Такие возможности процессора позволяют свести к минимуму потери времени при выполнении переходов и подключений процедур.

A.4.2. Регистры состояния и управления

Процессоры семейства C5x имеют четыре 16-разрядных регистра состояния и управления. Регистры ST0 и ST1 содержат информацию о состоянии, разнообразные условия и режимы, совместимые с C25, в то время как регистры PMST и CBCR содержат расширенную статусную и управляющую информацию для управления дополнительными возможностями ядра C5x. Эти регистры могут быть сохранены в памяти данных и загружены из памяти данных, что позволяет сохранять и восстанавливать состояние машины при входе и выходе из подпрограмм. ST0, ST1 и PMST имеют одноуровневый стек для автоматического сохранения контекста при обработке прерываний. Регистры автоматически восстанавливаются при возврате из прерывания, при этом не записывается и не восстанавливается бит XF.

Регистры PMST и CBCR отражены на память в странице 0. Таким образом, они могут участвовать в операциях в CALU или PLU. Они могут быть сохранены в

любом другом месте. Следует отметить, что команды с CALU и PLU меняют биты в этих регистрах в процессе фазы выполнения конвейера, следовательно, несколько последующих команд могут выполняться некорректно.

Команда LST загружает регистры состояния ST0 и ST1, а команда SST сохраняет эти регистры в памяти (INTM бит не изменяется). В отличие от регистров PMST и CBCR регистры состояния ST0 и ST1 не отражены на память и не могут участвовать в операциях с CALU и PLU. Определенные биты в этих регистрах могут быть изменены командами установки и сброса битов SETC и CLRC.

Рис. А.3 показывает внутреннюю организацию регистров. Некоторые биты в регистрах состояния являются зарезервированными. Табл. А.4 определяет назначение всех полей в этих регистрах.

А.4.3. Организация циклов. Конвейер

Для организации повторения одиночной команды в процессорах С5х используется 16-разрядный счетчик RPTC. В счетчик программно командой RPT загружается число N повторений. После загрузки счетчика, следующая одиночная команда будет выполнена N+1 раз. Не повторяются ряд арифметических команд и команд переходов, возвратов и вызовов подпрограмм. Повтор одиночной команды выгодно использовать при пересылках блоков, умножении матриц и т. п.

В архитектуре процессора предусмотрена также возможность организации циклического выполнения участка кода программы (организация циклов FOR и DO). Эта функция управляется через три регистра (PASR,PAER и BRCR) и флажок BRAF в регистре PMST. При выполнении команды RPTB адрес команды, непосредственно следующей за RPTB, загружается в регистр PASR. Из длинного непосредственного операнда, следующего за командным словом RPTB, загружается регистр PAER.

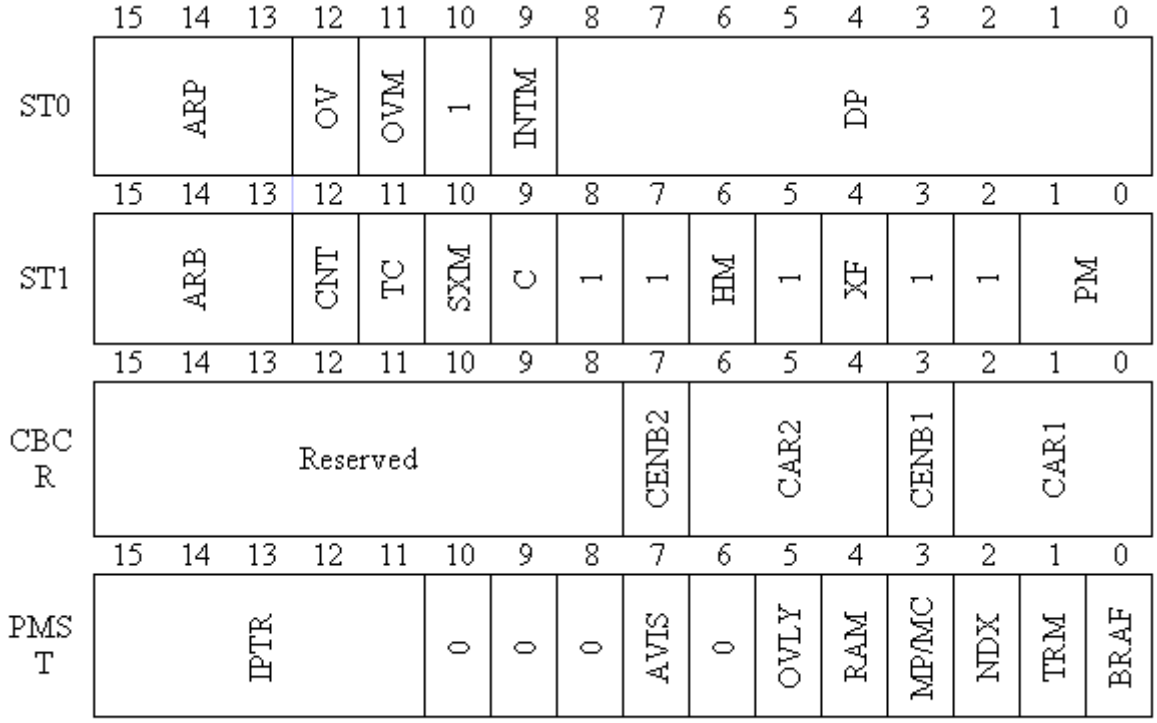

Рис. А.3. Организация регистров состояния и управления

Таблица А.4

Назначение полей в регистрах состояния и управления

Поле	Функция
ARB	Буфер указателя текущего вспомогательного регистра
ARP	Указатель текущего (активного) вспомогательного регистра
AVIS	Бит режима видимости адреса (если 0 – при обращении к внутренней памяти адрес выводится на внешние контакты)
BRAF	Бит активности выполнения повторяющихся блоков кодов
C	Бит переноса. Устанавливается в 1, если в результате суммирования генерируется перенос; сбрасывается в 0 , если при вычитании возникает заем
CAR1	Биты идентификации вспомогательного регистра, задействованного для первого циклического буфера
CAR2	Биты идентификации вспомогательного регистра, задействованного для второго циклического буфера
CENB1	Бит разрешения работы первого буфера. При CENB1 = 1 работа рарешена
CENB2	Бит разрешения работы второго буфера. При CENB2 = 1 работа рарешена
CNF	Бит управления конфигурацией внутреннего ОЗУ. При CNF = 1 блок B0 DARAM отражается в память программ, при CNF = 0 – в память данных

DP	Указатель текущей (активной) страницы памяти данных
HM	Бит режима HOLD. При HM = 1 – процессор останавливает работу, при HM = 0 – продолжает выполнение команд
INTM	Бит маскирования прерываний. При INTM = 0 – все маскируемые прерывания разрешены, при INTM = 1 – запрещены
IPTR	Указатель размещения таблицы векторов прерывания. При сбросе равен 0
MP/MC	Бит режима использования памяти микропроцессор/микрокомпьютер. При MP/MC = 0 внутреннее ПЗУ разрешено, при MP/MC = 1 – запрещено
NDX	Бит разрешения/запрещения дополнительного индексного регистра INDX. При сбросе в 0 этого бита обеспечивается режим совместимости с C2x (в качестве индексного используется регистр AR0)

Продолжение табл. А.4

Поле	Функция
OV	Бит индикации переполнения
OVLY	Бит режима использования памяти. При установке в 1 внутреннее ОЗУ (SARAM) разрешено для использования в качестве памяти данных
OVM	Бит режима переполнения. При OVM = 0 – нормальное переполнение, при OVM =1 – нет переполнения (режим ограничения)
PM	Бит режима работы сдвигателя при передаче PREG –> ACC
RAM	Бит режима использования памяти. При RAM = 1 разрешено использование внутренней SARAM в качестве памяти программ
SXM	Бит режима расширения знака
TC	Бит тест/управление. Используется при различных проверках. Есть команды перехода по значению этого бита
TRM	Бит разрешения множества T регистров (TREGs). При TRM = 0 – режим, совместимый с C2x, при TRM = 1 – режим C5x
XF	Бит состояния внешнего вывода. Индицирует состояние внешнего контакта XF

Следует отметить, что в теле цикла должно быть не менее трех команд, что связано с наличием конвейера. Выполнение команды RPTB автоматически устанавливает флаг BRAF в регистре PMST. При каждой модификации программного счетчика его содержимое сравнивается с содержимым регистра

PAER. Если равенство выполняется, содержимое регистра BRCR сравнивается с нулем. Если содержимое регистра BRCR больше нуля, то программный адрес из регистра PASR загружается в PC, содержимое BRCR декрементируется и начинается новое выполнение тела цикла. Если нет, загрузка программного счетчика не производится и флаг BRAF сбрасывается.

Для организации вложенных циклов необходимо сохранять содержимое регистров PASR, PAER и BRCR в памяти.

При написании программ для C5x следует иметь в виду, что выполнение любой команды в процессоре разбито на четыре фазы (выборка команды, декодирование, выборка операндов, выполнение), причем, может обрабатываться одновременно на разных фазах до трех инструкций. При этом следует иметь в виду, что команды модификации адресов, переконфигурирования процессора и пр. могут в определенных случаях выполняться некорректно.

A.5. Система прерываний

Ядро CPU C5x поддерживает 16 маскируемых пользователем прерываний. При этом конкретный процессор семейства C5x поддерживает не все 16. Например, C50 использует только девять из этих прерываний. Прерывания могут генерироваться последовательными портами (RINT, XINT, TRNT, TXNT), таймером (TINT), командами программного прерывания (TRAP, INTR). Сигнал RS имеет наивысший приоритет, INT16 – низший. Все сигналы прерывания имеют активный низкий уровень.

Расположение векторов прерываний и их приоритеты показаны в табл. A.5. Приоритет не установлен для команды TRAP (используемой для программного прерывания).

Вектора прерываний расположены в памяти с шагом в две ячейки, необходимые для хранения команды перехода. Расположение таблицы векторов прерываний в памяти определяется 5-разрядным полем IPTR в регистре PMST. Поле IPTR сбрасывается в ноль при сбросе (сигнал RS) , при этом вектора располагаются по адресу 0000h в пространстве программной памяти. Программный адрес таблицы векторов может быть изменен на начало любого из 32 блоков программной памяти размером в 2K 16-разрядных слов. Для этого необходимо записать номер блока в IPTR.

Таблица A.5

Расположение и приоритет векторов прерываний

Имя	Расположение (Hex)	Приоритет	Функция
RS	0	1	Внешний сигнал сброса

NMI	24	2	Немаскируемое прерывание
INT1	2	3	Внешнее пользовательское прерывание 1
INT2	4	4	Внешнее пользовательское прерывание 2
INT3	6	5	Внешнее пользовательское прерывание 3
TINT	8	6	Прерывание от внутреннего таймера
RINT	A	7	Прерывание приемника последовательного порта
XINT	C	8	Прерывание передатчика последовательного порта
TRNT	E	9	Прерывание приемника TDM порта
TXNT	10	10	Прерывание передатчика TDM порта
INT4	12	11	Внешнее пользовательское прерывание 4
–	14–21	–	Зарезервировано
TRAP	22	–	Команда TRAP
–	26–27	–	Зарезервировано
–	28–3F	–	Программные прерывания

Когда приходит запрос на прерывание, до обработки он сохраняется в 16-разрядном регистре флажков прерываний IFR.

Процессоры семейства C5x имеют расположенный в памяти регистр маски прерываний IMR для маскирования внутренних и внешних прерываний. На рис. А.4 показана внутренняя организация этого регистра. Запись единицы в разряд этого регистра разрешает прерывание при условии, что бит глобального разрешения маскируемых прерываний INTM (регистр ST0) сброшен в ноль.

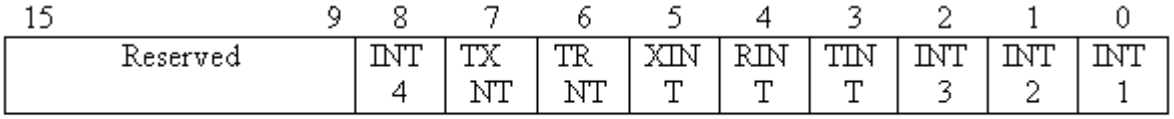

Рис. А.4. Внутренняя организация регистра IMR

При переходе к подпрограмме обработки прерывания автоматически сохраняется содержимое следующих регистров:

- ACC аккумулятор;
- ACCB буфер аккумулятора;
- PREG регистр результата умножения;
- ST0 регистр статуса 1;
- ST1 регистр статуса 2;
- PMST статусный регистр режима процессора;

- TREG0 временный регистр для умножителя;
- TREG1 временный регистр для счетчика сдвигателя;
- TREG2 временный регистр для битовых операций;
- INDX индексный регистр косвенной адресации;
- ARCR регистр сравнения дополнительного регистра.

A.6. Последовательный порт

Устройства семейства C5x имеют встроенный быстродействующий синхронный дуплексный последовательный порт ввода/вывода. Этот порт способен выполнять операции с частотой до 1/4 частоты машинного такта (CLKOUT1). Приемник и передатчик имеют двойную буферизацию и индивидуально управляются маскируемыми внешними сигналами прерывания. Кадры передачи могут иметь длину, равную байту или слову.

Порт делает возможной связь с последовательными устройствами, такими как кодеки, последовательные А/Ц преобразователи и др. Последовательный порт может также использоваться для связи с другими процессорами в мультипроцессорных приложениях (для этой цели более оптимизирован TDM порт).

Последовательный порт полностью статичный и может работать с любой низкой частотой, при этом максимальная частота

CLKOUT1/4 – F = 20 МГц.

Максимальная скорость передачи – 7,14 Мбит/с.

После сброса устройство находится в состоянии пониженного энергопотребления.

В табл. А.6 приводится назначение всех контактов последовательного порта.

Таблица А.6

Контакты последовательного порта

Контакт	Назначение
CLKX	Сигнал синхронизации передатчика
CLKR	Сигнал синхронизации приемника
DX	Линия передачи данных
DR	Линия приема данных
FSX	Фреймовый сигнал синхронизации передатчика
FSR	Фреймовый сигнал синхронизации приемника

Операции с последовательным портом производятся через три расположенных в памяти регистра (SPC, DXR, DRR) и два других регистра (XSR и RSR). Эти пять регистров перечислены в табл. А.7.

Таблица А.7

Регистры последовательного порта

Регистр	Назначение
SPC	Регистр управления последовательного порта
DXR	Регистр данных передатчика
DRR	Регистр данных приемника
XSR	Сдвиговый регистр передатчика
RSR	Сдвиговый регистр приемника

Регистр SPC управляет операциями с последовательным портом. Поля регистра SPC перечислены в табл. А.8.

Таблица А.8

Назначение полей регистра управления последовательным портом SPC

Бит	Имя поля	Функция
0	Reserved	Зарезервировано, при чтении регистра равен 0
1	DLB	Бит циклического режима. В циклическом режиме (DLB = 1) контакт DR коммутируется с контактом DX, контакт FSR – с контактом FSX и контакт CLKR – с контактом CLKX. При установке режима должно соблюдаться условие TXM = 1. Используется для тестирования последовательного порта
2	FO	Бит управления длиной передаваемого/принимаемого слова, при FO = 1 длина слова 8 разрядов, при FO = 0 – 16 разрядов
3	FSM	Бит управления режимом кадровой синхронизации. При FSM = 1 требуется сигнал синхронизации для начала приема/пересылки каждого слова (режим передачи одиночных слов), при FSM = 0 сигнал требуется только при начале приема/передачи массива слов (продолжительный режим)
4	MCM	Бит режима синхронизации. Если MCM = 0, сигнал синхронизации передатчика берется с внешнего контакта, если MCM = 1 – с внутреннего источника синхронизации (CLKOUT1/4)
5	TXM	Бит режима передачи. Если TXM = 1, контакт FSX конфигурируется на выход, если TXM = 0 – на вход

6 7	XRST RRSR	Сигналы сброса передатчика и приемника. Сбрасывают передатчик и приемник соответственно. Если SPC модифицируется для реконфигурирования порта, должны быть сделаны две записи в SPC. При первой загрузке в эти разряды должны быть записаны нули и определены разряды конфигурации 1–5. При второй записи в эти разряды должны быть записаны единицы. После второй записи осуществляется сброс. Когда XRST = 0, RRST = 0, MCM = 0, от порта отключается внутренняя синхронизация и схема работает в режиме пониженного энергопотребления

Окончание табл. А.8

Назначение полей регистра управления последовательным портом SPC

Бит	Имя поля	Функция
8 9	IN0 IN1	Биты, повторяющие текущий уровень на контактах CLKR и CLKX соответственно
10 11	RRDY XRDY	Флаги готовности приемника и передатчика соответственно. Флаг RRDY показывает, что пересылка из регистра RSR в DRR совершена. Флаг XRDY показывает, что пересылка из регистра DXR в регистр XSR совершена. После осуществления пересылок генерируются соответствующие прерывания
12	XSREMPTY	Флаг “передатчик пуст” (активный низкий уровень). Устанавливается, если сдвиговый регистр пуст, а в регистр данных не произведено записи
13	RSRFULL	Флаг “приемник полон”. Устанавливается если сдвиговый регистр принял данные, а из регистра данных не произведено чтение
14 15	SOFT FREE	Биты эмуляции для отладчика высокого уровня. С помощью этих бит устанавливается режим работы последовательного порта при программной точке останова. Возможна установка трех режимов – немедленный останов, останов после завершения приема/передачи слова, свободное выполнение

Приложение В

РЕГИСТРЫ СОСТОЯНИЯ И УПРАВЛЕНИЯ ПРОЦЕССОРА

Процессоры семейства C5x имеют четыре регистра состояния и управления. Регистры ST0 и ST1 содержат информацию состояния, разнообразные условия и режимы, совместимые с C25, в то время как регистры PMST и CBCR содержат расширенную управляющую информацию и информацию состояния для управления дополнительными возможностями C5x. Содержание всех регистров может быть сохранено в памяти данных и загружено из памяти данных, что позволяет сохранять и восстанавливать состояние машины при входе и выходе из подпрограмм.

Регистры ST0, ST1 и PMST имеют собственный одноуровневый стек для автоматического сохранения контекста при обработке прерываний. Регистры автоматически восстанавливаются при возврате из прерывания, при этом не записывается и не восстанавливается бит XF.

Регистры PMST и CBCR находятся в памяти данных на странице 0. Таким образом, они могут участвовать в операциях в CALU или PLU. Их содержимое может быть сохранено в любом другом месте. Следует отметить, что команды с CALU и PLU меняют биты в этих регистрах в начальной фазе выполнения конвейера, следовательно, несколько последующих команд могут выполняться некорректно. Для того чтобы избежать ошибки при выполнении последующих операций, после работы с CALU или PLU регистры PMST и CBCR следует перезагрузить.

Команда LST загружает регистры состояния ST0 и ST1, а команда SST сохраняет эти регистры в памяти (INTM бит не изменяется). В отличие от регистров PMST и CBCR, регистры состояния ST0 и ST1 не отражены на память и не могут участвовать в операциях с CALU и PLU. Определенные биты в этих регистрах могут быть изменены командами установки и сброса битов SETC и CLRC.

Рис. В.1 показывает распределение признаков и опций по разрядам регистров. Некоторые биты в регистрах состояния являются зарезервированными. Назначение полей в этих регистрах приведено в табл. А.4. приложения А.

ST0:

15 14 13	12	11	10	9	8 7 6 5 4 3 2 1 0
ARP	OV	OVM	1	INTM	DP

ST1:

15 14 13	12	11	10	9	8 7	6	5 4	3 2	1 0
ARB	CNF	TC	SXM	C	1 1	HM	1 XF	1 1	PM

CBCR:

15 14 13 12 11 10 9 8	7	6 5 4	3	2 1 0
Reserved	CENB2	CAR2	CENB1	CAR1

PMST:

15 14 13 12 11	10 9 8	7	6	5	4	3	2	1	0
IPTR	0 0 0	AVIS	0	OVLYY	RAM	MP/MC	NDX	TRM	BRAF

Рис. В.1. Организация признаков и опций по разрядам регистров состояния и управления

Приложение С

ТАЙМЕР ЦПОС TMS320C50

Таймер представляет собой программно-управляемый делитель частоты и может быть использован для:

- формирования опорных частот для внешних устройств, например, для схемы АЦП/ЦАП;
- формирования временных отметок, например, для реализации прерываний.

На рис. С.1. представлена логическая схема таймера. Таймер представляет собой последовательное соединение двух вычитающих счетчиков – 16-разрядного счетчика TIM и 4-разрядного счетчика PSC. За каждый период частоты CLKOUT1 (частоты работы процессора) длительностью 50 нс из счетчика PSC вычитается единица и при значении содержимого PSC, равном нулю, соответственно происходит вычитание в счетчике TIM.

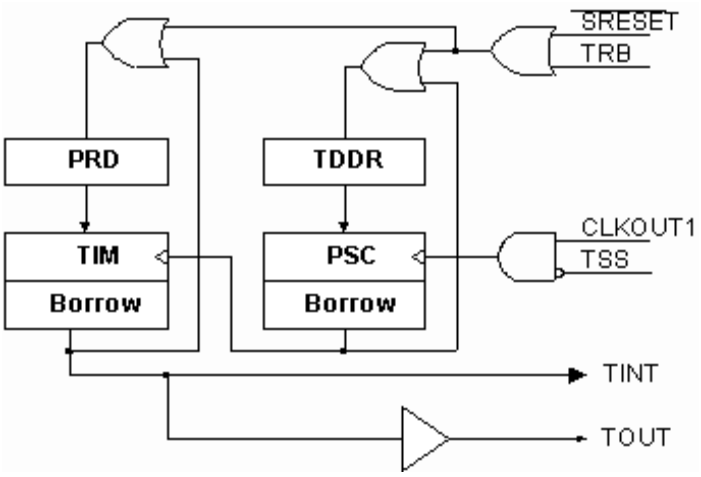

Рис. С.1. Структурная схема таймера

Как видно из рис. С.1, частота генерации прерываний TINT и выдаваемого на выход сигнала TOUT, равняется частоте сигнала CLKOUT1, проходящего через две независимые цепи деления частоты. Каждая из цепей состоит из вычитающего счетчика и регистра периода. Значения счетчика и регистра периода первой цепи – это значения полей PSC и TDDR регистра TCR (табл. С.1). Во второй цепи счетчик и регистр периода представлены отраженными на память, 16-разрядными регистрами TIM и PRD. Каждый раз, когда содержимое счетчика TIM уменьшается до нуля, генерируется сигнал TINT и происходит загрузка в счетчик содержимого своего регистра периода. Сгенерированный сигнал прерывания с выхода второй цепи деления частоты посылается в процессор и поступает на вывод (TOUT).

Частота генерации прерываний, идущих от таймера, определяется по следующей формуле:

$$TINT = 1/(tc(c) \times (TDDR + 1) \times (PRD + 1)), \quad (C.1)$$

где tc(c) – период CLKOUT1;

TDDR – содержимое регистра TDDR;

PRD – содержимое регистра PRD.

Максимальный коэффициент деления таймера равен $(2^{16}+1) \times (2^4+1)$.

Управление работой таймера осуществляется с помощью регистра управления TCR. Останов таймера выполняется путем записи 1 в бит TSS. Для того чтобы снова запустить таймер, следует записать 0 в ячейку TSS. При приходе сигнала сброса (при записи 1 в бит TRB) бит TSS становится равным 0, и таймер сразу начинает работу. Для того чтобы перезагрузить таймер, также записывают 1 в ячейку TRB (табл. С.1).

Таблица С.1

Назначение полей в регистре управления таймером

Бит	Название поля	Описание
0–3	TDDR	Определяет коэффициент деления работы процессора. Значение по умолчанию 0
4	TSS	Останов таймера TSS =1, старт таймера TSS = 0
5	TRB	Перегрузка и сброс таймера TRB = 1
6–9	PSC	Вычитающийсчетчик

Содержимое регистра PRD загружается в регистр счета TIM, когда содержимое счетчика уменьшается до нуля или при записи 1 в бит TRB регистра управления таймером. Так же, как и при работе пары регистров TIM/PRD, содержимое регистра TDDR не загружается сразу в масштабирующий счетчик PSC. Счетчик PSC загружается при уменьшении до нуля, а также при поступлении сигнала сброса. Содержимое счетчика PSC может быть прочитано при чтении регистра TCR, но не может быть загружено непосредственно из программы. В регистре управления таймером имеются специально зарезервированные биты эмуляции 10 и 11, которые позволяют запоминать состоянии таймера при установлении точек останова программы во время работы в среде программы-монитора.

Ниже представлен пример программы инициализации таймера, организации процедур запуска и останова таймера для организации прерываний и генерации на выходе частоты CLKOUT1/50000.

```
        .text                   ; директива ассемблера, начало секции
TimerInit:                      ; процедура инициализации таймера
```

```
        LDP     #0              ; установление нулевой страницы
                                ; памяти, где расположены регистры
                                ; управления работой таймера
        SPLK    #50000,PRD      ; загрузка регистра периода
        SPLK    #0Ah,IMR        ; разрешение прерывания TINT и
                                ; прерывания для программы-монитора
                                ;  INT2
        RET                     ; возврат из процедуры
StartTimer:                     ; процедура запуска таймера
        SPLK    #020h,TCR       ; перезагрузка и старт таймера
        RET                     ; возврат из процедуры
StopTimer:                      ; процедура остановки таймера
        SPLK    #10h            ; запись 1 в 4-й бит (TSS) регистра (TCR)
        RET                     ; возврат из процедуры
```

Приложение D

Краткий обзор команд процессора TMS320C50

В таблице использованы следующие сокращения и обозначения:

количество слов – длина команды и количество ячеек памяти, занимаемое командой;

количество циклов – количество командных циклов, затрачиваемых на выполнение команды;

(ACC) – содержимое аккумулятора.

Таблица D.1

СПИСОК КОМАНД

Мнемоника	Описание	Кол-во слов	Количество циклов
Команды, связанные с использованием аккумулятора			
ABS	Взятие модуля от содержимого аккумулятора	1	1
ADCB	Сложение (ACCB) + (ACC) с переносом	1	1
ADD	Сложение с содержимым аккумулятора	1/2	1 (операнд – 8 бит) 2 (операнд – 16 бит)
ADDB	Сложение (ACCB) + (ACC)	1	1
ADDC	Сложение с (ACC) с переносом	1	1
ADDS	Сложение с младшим словом содержимого аккумулятора с подавлением знакового расширения	1	1
ADDT	Сложение с содержимым аккумулятора со сдвигом, определенным в TREG1	1	1

AND	Логическая операция И с (ACC)	1/2	1 (операнд – 8 бит) 2 (операнд –16 бит)
ANDB	Логическая операция (ACCB) И (ACC)	1	1
BSAR	Циклический сдвиг (ACC) вправо	1	1
CMPL	Логическая инверсия (ACC)	1	1
CRGT	Проверка (ACCB) > (ACC)	1	1
CRLT	Проверка (ACCB) < (ACC)	1	1
EXAR	Обмен (ACCB) ↔ (ACC)	1	1
LACB	Загрузка (ACCB) в (ACC)	1	1
LACC	Загрузка аккумулятора со сдвигом	1/2	1 (операнд – 8 бит) 2 (операнд –16 бит)
LACL	Загрузка младшего слова аккумулятора	1	1
LACT	Загрузка аккумулятора со сдвигом, определенным в TREG1	1	1
LAMM	Загрузка аккумулятора содержимым регистра, расположенного в памяти	1	1 (регистр процессора) 2 (регистр периферии)
NEG	Арифметическая инверсия (ACC)	1	1
NORM	Нормализация содержимого аккумулятора	1	1
OR	Логическая операция ИЛИ с (ACC)	1/2	1 (операнд – 8 бит) 2 (операнд –16 бит)

Продолжение табл. D.1

Мнемоника	Описание	Кол-во слов	Количество циклов
ORB	Логическая операция ИЛИ (ACCB) и (ACC)	1	1
ROL	Сдвиг содержимого аккумулятора влево	1	1
ROLB	Сдвиг (ACC) и (ACCB) влево	1	1
ROR	Сдвиг (ACC) вправо	1	1
RORB	Сдвиг (ACC) и (ACCB) вправо	1	1
SACB	Сохранение (ACC) в (ACCB)	1	1
SACH	Сохранение старшего слова (ACC) со сдвигом	1	1
SACL	Сохранение младшего слова (ACC) со сдвигом	1	1
SAMM	Сохранение (ACC) в регистре, размещенный в памяти	1	1 (регистр процессора) 2 (регистр периферии)
SATH	Циклический сдвиг (ACC) на 16 разрядов вправо, согласно TREG1	1	1
SATL	Циклический сдвиг (ACC) вправо, согласно TREG1	1	1
SBB	Вычитание (ACCB) из (ACC)	1	1
SBBB	Вычитание (ACCB) из (ACC) с заемом	1	1
SFL	Сдвиг (ACC) влево	1	1
SFLB	Сдвиг (ACC) и (ACCB) влево	1	1
SFR	Сдвиг (ACC) вправо	1	1
SFRB	Сдвиг (ACC) и (ACCB) вправо	1	1
SUB	Вычитание из (ACC)	1/2	1 (операнд – 8

			бит) 2 (операнд –16 бит)
SUBB	Вычитание из (ACC) с заемом	1	1
SUBC	Условное вычитание из (ACC)	1	1
SUBS	Вычитание из (ACC) с подавлением знакового расширения	1	1
SUBT	Вычитание из (ACC) со сдвигом, определенным в TREG1	1	1
XOR	Логическая операция исключающее ИЛИ с (ACC)	1/2	1 (операнд – 8 бит) 2 (операнд –16 бит)
XORB	Логическая операция (ACCB) ⊕ (ACC)	1	1
ZALR	Обнуление (ACC) и загрузка старшего слова (ACC) с округлением	1	1
ZAP	Обнуление содержимого аккумулятора и P-регистра	1	1
Команды, связанные с использованием вспомогательных регистров и указателя страниц памяти			
ADRK	Сложение (ARn) с константой	1	1
CMPR	Сравнение (ARn) с (ARCR)	1	1
LAR	Загрузка вспомогательного регистра	1/2	2
LDP	Установка текущего номера страницы	1	2

Продолжение табл. D.1

Мнемоника	Описание	Кол-во слов	Количество циклов
MAR	Установка текущего вспомогательного	1	1

	регистра		
SAR	Сохранение содержимого вспомогательного регистра	1	1
SBRK	Вычитание из содержимого вспомогательного регистра константы	1	1
Команды параллельного логического устройства			
APL	Логическое И содержимого ячейки памяти и содержимого DMBR (либо константы)	1/2	1 (второй операнд – DMBR) 2 (второй операнд – константа 16 бит)
CPL	Сравнение содержимого ячейки памяти и содержимого DMBR (либо константы)	1/2	1 (второй операнд –DMBR) 2 (второй операнд – константа 16 бит)
OPL	Логическое ИЛИ содержимого ячейки памяти и содержимого DMBR (либо константы)	1/2	1 (второй операнд – DMBR) 2 (второй операнд – константа 16 бит)
SPLK	Сохранение константы в ячейке памяти	2	2
XPL	Логическое исключающее ИЛИ содержимого ячейки памяти и содержимого DMBR (либо константы)	1/2	1 (второй операнд – DMBR) 2 (второй операнд – константа 16 бит)
Команды, связанные с умножением			
APAC	Сложение содержимого аккумулятора и P-регистра	1	1
LPH	Загрузка старшего слова P-регистра	1	1
LT	Загрузка TREG0	1	1

LTA	Загрузка TREG0 и аккумуляция предыдущего произведения	1	1
LTD	Загрузка TREG0, аккумуляция предыдущего произведения и сдвиг данных	1	1
LTP	Загрузка TREG0 и сохранение предыдущего произведения в аккумуляторе	1	1
LTS	Загрузка TREG0 и вычитание предыдущего произведения	1	1
MAC	Умножение и аккумуляция предыдущего произведения	2	3
MACD	Умножение, аккумуляция предыдущего произведения и сдвиг данных	2	3
MADD	Умножение и аккумуляция предыдущего произведения с множителем, адресуемым BMAR	1	3

Продолжение табл. D.1

Мнемоника	Описание	Кол-во слов	Количество циклов
MADS	Умножение и аккумуляция предыдущего произведения с множителем, адресуемым BMAR и сдвигом данных	1	3
MPY	Умножение	1/2	1 (операнд – 8 бит) 2 (операнд –16 бит)
MPYA	Умножение и аккумуляция предыдущего произведения	1	1
MPYS	Умножение и вычитание предыдущего произведения	1	1
MPYU	Беззнаковое умножение	1	1

PAC	Загрузка аккумулятора содержимым P-регистра	1	1
SPAC	Вычитание из аккумулятора содержимого P-регистра	1	1
SPH	Сохранение старшего слова P-регистра	1	1
SPL	Сохранение младшего слова P-регистра	1	1
SPM	Установка режима сдвига произведения при его передаче	1	1
SQRA	Возведение в квадрат и аккумуляция предыдущего произведения	1	1
SQRS	Возведение в квадрат и вычитание предыдущего произведения	1	1
ZPR	Обнуление P-регистра	1	1
Команды переходов			
B[D]	Безусловный переход	2	4 (2 – при задержке)
BACC[D]	Переход на адрес, указываемый аккумулятором	1	4 (2 – при задержке)
BANZ[D]	Переход при ARn > 0	2	4 (условие выполняется; 2 – при задержке) 2 (условие не выполнено)
BCND[D]	Условный переход	2	4 (условие выполняется; 2 – при задержке) 2 (условие не выполнено)
CALA[D]	Вызов подпрограммы по адресу в аккумуляторе	1	4 (2 – при задержке)

CALL[D]	Вызов подпрограммы	2	4 (2 – при задержке)
CC[D]	Условный вызов подпрограммы	2	4 (условие выполняется; 2 – при задержке) 2 (условие не выполнено)
INTR	Вызов программного прерывания	1	4
NMI	Вызов немаскируемого прерывания	1	4
RET[D]	Возврат из подпрограммы	1	4 (2 – при задержке)
RETC[D]	Условный возврат из подпрограммы	1	4 (условие выполняется; 2 – при задержке) 2 (условие не выполнено)

Окончание табл. D.1

Мнемоника	Описание	Кол-во слов	Количество циклов
RETE	Возврат из процедуры обработки прерывания и разрешение прерываний	1	4
RETI	Возврат из процедуры обработки прерывания	1	4
TRAP	Вызов прерывания TRAP	1	4
XC	Условное исполнение 1–2 следующих инструкций	1	1
Команды ввода/вывода и блоковых операций			
BLDD	Блочное перемещение данных внутри памяти данных	1/2	2 (операнд – BMAR)

			3 (операнд – константа 16 бит)
BLDP	Блочное перемещение данных из памяти данных в память программ	1	2
BLPD	Блочное перемещение данных из памяти программ в память данных	1/2	2 (операнд – BMAR) 3 (операнд – константа 16 бит)
DMOV	Сдвиг данных в памяти данных	1	1
IN	Ввод данных из порта	2	2
LMMR	Загрузка регистра, расположенного в памяти	2	2 (регистр процессора) 3 (регистр периферии)
OUT	Вывод данных в порт	2	3
SMMR	Сохранение содержимого регистра, расположенного в памяти	2	2 (регистр процессора) 3 (регистр периферии)
TBLR	Перемещение данных из памяти программ в память данных	1	3
TBLW	Перемещение данных из памяти данных в память программ	1	3
Команды управления			
BIT	Битовая проверка	1	1
BITT	Проверка бита, указываемого TREG2	1	1
CLRC	Очистка контрольного бита TC	1	1
IDLE	Ожидание прихода прерывания	1	1
IDLE2	Ожидание прихода прерывания –	1	1

	режим с понижением питания		
LST	Загрузка контрольного регистра	1	2
NOP	Нет операции	1	1
POP	Загрузка вершины стека в аккумулятор	1	1
POPD	Загрузка вершины стека в ячейку памяти	1	1
PSHD	Помещение содержимого ячейки памяти в стек	1	1
PUSH	Помещение содержимого аккумулятора в стек	1	1
RPT	Повторение следующей команды	1/2	2
RPTB	Повторение блока команд	2	2
RPTZ	Обнуление аккумулятора и повторение следующей команды	2	2
SETC	Установка контрольного бита	1	1
SST	Сохранение регистра состояния	1	1

Литература

1. TMS320C5x User’s Guide. Digital Signal Processing Products. Texas Instruments Incorporated, 1995.
2. TMS320C5x Sourser Debugger Users Guide. Texas Instruments Incorporated, 1994.
3. TMS320C1x/C2x/C2xx/C5x Assembly Language Tools. User’s Guide. Microprocessor Development Systems. Texas Instruments Incorporated, 1995.
4. Солонина А.И., Улахович Д.А., Яковлев Л.А. Алгоритмы и процессоры обработки сигналов. – СПб.: БХВ-Петербург, 2001.