МИКРОСХЕМЫ ИНТЕГРАЛЬНЫЕ

1867ВЦ2Т, 1867ВЦ2АТ

Руководство пользователя

2005 г.

Содержание

1 Введение

В настоящее время цифровая обработка сигналов (ЦОС) охватывает широкий спектр практических приложений. В качестве примеров можно назвать цифровую фильтрацию, обработку изображений, быстрое преобразование Фурье (БПФ), телекоммуникацию и цифровую звукотехнику. Как в этих, так и в других областях цифровая обработка сигналов имеет общие особенности:

- большой объем вычислений,
- работа в реальном времени,
- обработка оцифрованных данных,
- гибкость системы ЦОС.

Развитие техники интегральных схем позволило создать более быстрые микропроцессоры и микро-ЭВМ. В результате во многих областях, связанных с цифровой обработкой сигналов, матричные процессоры стали вытесняться разрядномодульными микропроцессорными секциями, а затем однокристальными микро-ЭВМ. Цена процессоров цифровой обработки сигналов (ПЦОС) резко снизилась, открыв дорогу для их самого широкого внедрения. Возросшая производительность СБИС расширила область применения ЦПОС, по традиции ограничивавшуюся телефонной связью, и ЦПОС стали применяться в графических системах и системах обработки изображений, а затем и в звукотехнике.

Одним из основных достижений техники ЦОС стали однокристальные цифровые процессоры сигналов, примером которых служат процессоры серии TMS320.

К настоящему время разработаны отечественные аналоги процессоров цифровой обработки сигналов серии TMS320:

- М1867ВМ1 (функционально совместимый TMS320C10);

- 1867ВМ2 (функционально совместимый TMS320C25);

- 1867ВЦ1Ф (функционально совместимый TMS320C30).

Настоящее руководство пользователя посвящено 16-разрядному процессору ЦОС с фиксированной запятой 1867ВЦ2Т (1867ВЦ2АТ), который является функциональным аналогом микросхемы TMS320C50.

2 Назначение и основные технические характеристики микросхем

Сочетание модифицированной Гарвардской архитектуры (раздельные шины для памяти программ и памяти данных с возможностью межшинного обмена), дополнительных внутрикристальных периферийных устройств, большого объема внутрикристальной памяти и более высокой специализации системы команд – основа операционной гибкости и производительности ПЦОС 1867ВЦ2Т (1867ВЦ2АТ). Также реализована совместимость (на уровне исходных кодов) программ процессора 1867ВЦ2Т (1867ВЦ2АТ) с М1867ВМ1 и 1867ВМ2.

2.1 Архитектурные характеристики микропроцессора

Процессор 1867ВЦ2Т (1867ВЦ2АТ) – аналог TMS320C50, представителя пятого поколения семейства TMS320. Эти процессоры цифровой обработки сигналов (ПЦОС) производятся по статической КМОП технологии интегральных схем.

Формат представления данных		фиксированная запятая
Цикл выполнения команд, нс	1867ВЦ2Т	50/25
	1867ВЦ2АТ	35/17,5
Количество команд		124
Умножитель, бит		16 х 16
Разрядность АЛУ, бит		32
Объем ОЗУ данных, бит		10К × 16
Объем ПЗУ программ, бит		2К × 16
Объем адресуемой внешней памяти, бит		224К × 16
Таймер 16-разрядный		1
Последовательный порт		1
Мультиплексный последовательный порт с разделением по времени		1
Тактовая частота, МГц	1867ВЦ2Т	40
	1867ВЦ2АТ	57
Диапазон температур окружающей среды, °С		- 60 ÷ +85
Напряжение питания, В		5 ± 0,5
Технология изготовления		КМОП

Ниже приведена сравнительная таблица 2.1 процессоров ЦОС 1867ВЦ2Т, 1867ВЦ2АТ разработки ФГУП "НИИЭТ".

Таблица 2.1 - Сравнительная характеристика процессоров ЦОС

	М1867ВМ1 («Тост»)	Л1867ВМ2 («Тибр»), 1867ВМ2 («Тибр П»)	1867ВЦ2Т , 1867ВЦ2АТ («Таврия-Т»)
Формат представления данных	ФЗ	ФЗ	ФЗ
Цикл выполнения команд, нс	200	100	50/25, 35/17,5
Умножитель, бит	16 × 16	16 × 16	16 × 16
Разрядность АЛУ, бит	32	32	32
Объем ОЗУ данных, бит	144 × 16	544 × 16	10К × 16
Объем ПЗУ программ, бит	1,5К × 16	4К × 16	2К × 16
Объем адресуемой внешней памяти, бит	4К × 16	64К × 16	224К × 16
Таймер 16-разрядный	----	1	1
Последовательный порт	----	1	1
Мультиплексный последовательный порт с разделением по времени	----	----	1
JTAG интерфейс тестирования и отладки	----	----	1
Тип корпуса	2123.40-6 (DIP)	4235/88-1 (QFP)	4229.132-3
Количество выводов, шт	40	88	132

2.2 Конструктивные характеристики микросхем

Микросхемы выполнены в корпусе 4229.132-3 («Монополия-132»).

Масса микросхем не более 12 г.

Герметизация микросхем осуществляется шовной роликовой сваркой. Показатель герметичности корпуса по скорости утечки гелия не более 5×10^{-3} Па·см^3/с.

Микросхемы не имеют собственных резонансных частот ниже 100 Гц.

Условное графическое обозначение ИМС приведено на рисунке 2.1.

Функциональное назначение выводов приведено в таблице 2.2.

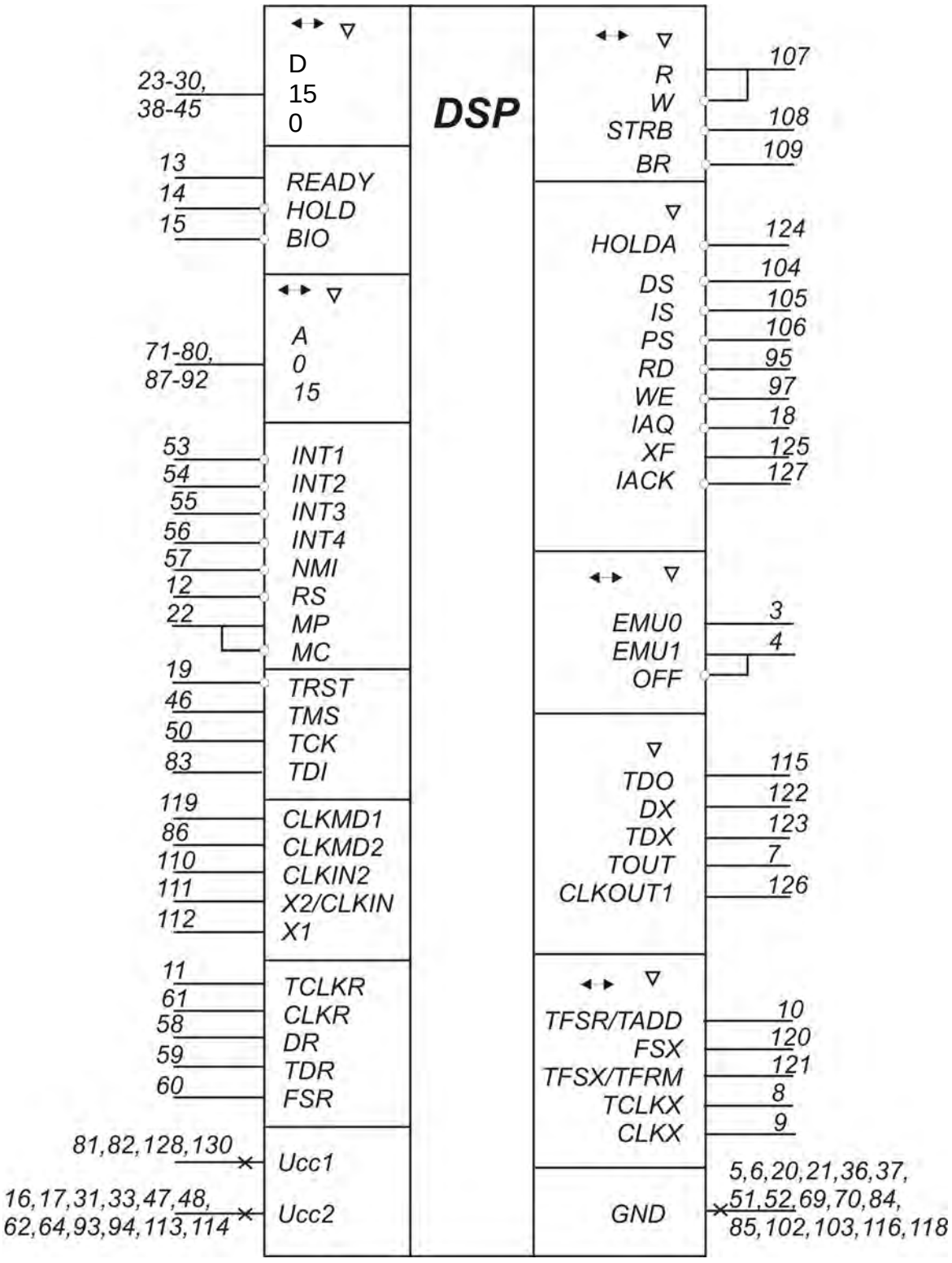


Рисунок 2.1 - Условное графическое обозначение ИМС 1867ВЦ2Т, 1867ВЦ2АТ

Таблица 2.2 - Функциональное назначение выводов микросхем

Номер вывода	Обозначение вывода	Функциональное назначение вывода
1	2	3
3	EMU0	вывод нулевого разряда эмулятора
4	EMU1/$\overline{\text{OFF}}$	вывод первого разряда эмулятора /перевод Z-выводов в высокоимпедансное состояние
5, 6	GND	шина "земли" выходных буферов
7	TOUT	выход таймера
8	TCLKX	синхросигнал передачи по последовательному порту с разделением по времени
9	CLKX	синхросигнал передачи по последовательному порту
10	TFSR/TADD	кадровый импульс синхронизации приема данных / адрес порта в режиме разделения по времени
11	TCLKR	синхросигнал приема по последовательному порту с разделением по времени
12	$\overline{\text{RS}}$	вход сброса
13	READY	вход готовности внешнего устройства
14	$\overline{\text{HOLD}}$	вход активизации состояния удержания
15	$\overline{\text{BIO}}$	вход управления инструкцией перехода
16, 17	U_{CC2}	шина питания выходных буферов
18	$\overline{\text{IAQ}}$	строб выполнения инструкции
19	$\overline{\text{TRST}}$	включение JTAG-тестирования
20, 21	GND	общая шина ("земля") ядра и входных буферов
22	MP / $\overline{\text{MC}}$	выбор режима микропроцессор/микрокомпьютер
23–30, 38-45	D15 - D0	шестнадцатиразрядная параллельная шина данных (вход/выход)
31, 33	U_{CC2}	питание выходных буферов шины данных
36, 37	GND	общая шина ("земля") выходных буферов
46	TMS	выбор режима JTAG-тестирования

Продолжение таблицы 2.2

1	2	3
47, 48	U_{CC2}	питание выходных буферов шины данных
50	TCK	тактовая синхронизация JTAG-тестирования
51, 52	GND	общая шина выходных буферов данных
53-56	$\overline{INT1} - \overline{INT4}$	входы внешних прерываний
57	$\overline{NMI}$	вход немаскируемого прерывания
58	DR	вход данных последовательного порта
59	TDR	вход данных последовательного порта с разделением по времени
60	FSR	старт-импульс приема данных по последовательному порту
61	CLKR	синхросигнал приема данных последовательного порта
62, 64	U_{CC2}	питание выходных буферов адресной шины
69, 70	GND	общая шина выходных буферов адреса
71-80, 87-92	A0 – A15	шестнадцатиразрядная параллельная шина адреса (вход/выход)
81, 82	U_{CC1}	шина питания ядра и входных буферов
83	TDI	вход данных JTAG-тестирования
84, 85	GND	общая шина ("земля") выходных буферов адреса
86	CLKMD2	вход выбора режима тактирования
93, 94	U_{CC2}	питание выходных буферов адресной шины
95	$\overline{RD}$	разрешение чтения
97	$\overline{WE}$	разрешение записи
102, 103	GND	общая шина выходных буферов управления
104	$\overline{DS}$	сигнал выбора пространства данных
105	$\overline{IS}$	сигнал выбора пространства ввода/вывода
106	$\overline{PS}$	сигнал выбора пространства программ
107	$R / \overline{W}$	сигнал чтения/записи
108	$\overline{STRB}$	сигнал стробирования, вход/выход

Продолжение таблицы 2.2

1	2	3
109	$\overline{BR}$	сигнал запроса шины данных, вход/выход
110	CLKIN2	вход внешнего генератора
111	X2/CLKIN	вход кварцевого резонатора / вход внешней синхронизации
112	X1	выход кварцевого резонатора
113, 114	U_{CC2}	шина питания выходных буферов
115	TDO	выход данных JTAG-тестирования
116, 118	GND	общая шина ("земля") ядра и входных буферов
119	CLKMD1	вход выбора режима тактирования
120	FSX	старт-импульс передачи по последовательному порту
121	TFSX/TFRM	старт-импульс передачи по последовательному порту с разделением по времени / фрейм данных
122	DX	выход данных последовательного порта
123	TDX	выход данных последовательного порта с разделением по времени
124	$\overline{HOLDA}$	сигнал подтверждения состояния удержания
125	XF	сигнал флага общего назначения
126	CLKOUT1	выход генератора машинного цикла
127	$\overline{IACK}$	сигнал подтверждения прерывания
128, 130	U_{CC1}	шина питания ядра и входных буферов
Примечание: Выводы 1, 2, 32, 34, 35, 49, 63, 65-68, 96, 98-101, 117, 129, 131, 132 не задействованы.		

2.3 Электрические характеристики микросхем

Электрические характеристики микросхем 1867ВЦ2Т, 1867ВЦ2АТ при приемке и поставке приведены в таблице 2.3.

Значения предельно-допустимых электрических режимов эксплуатации в диапазоне рабочих температур приведены в таблице 2.4.

Таблица 2.3 - Электрические параметры микросхем 1867ВЦ2Т, 1867ВЦ2АТ при приемке и поставке

Наименование параметра, единица измерения, режим измерения		Буквенное обозначение параметра	Норма параметра		Температура среды, °С
			не менее	не более	
1		2	3	4	5
1 Выходное напряжение низкого уровня, В I_{OL} = 2 мА; U_{CC} = 4,5 В		U_{OL}		0,6	–60 ± 3 25 ± 10 85 ± 3
2 Выходное напряжение высокого уровня, В I_{OH} = –0,3 мА; U_{CC} = 4,5 В		U_{OH}	2,4		–60 ± 3 25 ± 10 85 ± 3
3 Входной ток высокого уровня, мА U_{IH} = U_{CC} = 5,5 В	Вход $\overline{TRST}$	I_{IH}		800	
	Входы TMS, TCK, TDI			10	25 ± 10 85 ± 3
	Вход X2/CLKIN			50	
	Все остальные входы			10	
4 Входной ток низкого уровня, мА U_{IL} = 0 В	Вход $\overline{TRST}$	I_{IL}	–10		25 ± 10 85 ± 3
	Входы TMS, TCK, TDI		–500		
	Вход X2/CLKIN		–50		
	Все остальные входы		–10		
5 Выходной ток в состоянии "выключено", мкА U_{OL}= 0 В; U_{OH} = U_{CC} = 5,5 В		I_{OZH} I_{OZL}	– –20	20 –	25 ± 10 85 ± 3

Продолжение таблицы 2.3

1	2		3	4	5
6 Динамический ток потребления ядра, мА U_{CC} = 5,5 В; f_{CI} = 10 МГц	I_{OCC1}			50	–60 ± 3 25 ± 10 85 ± 3
7 Динамический ток потребления периферии, мА U_{CC} = 5,5 В; f_{CI} = 10 МГц	I_{OCC2}			40	–60 ± 3 25 ± 10 85 ± 3
8 Время переключения на выходе CLKOUT1, нс U_{CC} = 5,0 В; f_{CI} = 10 МГц	время нарастания	t_r		5	
	время спада	t_f		5	
9 Функциональный контроль, U_{CC} = (4,5÷5,5) В	ФК				

Примечания

1 Измерение I_{OCC1} и I_{OCC2} производится на тесте ФК "Start" при C_L = 50 пФ.

2 Выходное напряжение высокого уровня U_{OH} должно быть не менее (U_{CC}–0,7) В при токе I_{OH} = –0,3 мА и U_{CC} в диапазоне (4,5 ÷ 5,5) В.

3 Функциональный контроль (ФК) проводится в соответствии с "Программой и методикой функционального контроля" КФДЛ.431299.011 ПМ.

Для микросхем 1867ВЦ2Т ФК проводят на частоте 40 МГц, а для микросхем 1867ВЦ2ТА на частоте 57МГц.

Таблица 2.4 – Предельно-допустимые режимы эксплуатации микросхем в диапазоне рабочих температур от минус 60 ºС до плюс 85 ºС

Наименование параметра режима, единица измерения	Буквенное обозначение	Предельно-допустимый режим	
		не менее	не более
1	2	3	4
1 Напряжение источника питания, В	U_{CC}	4,5	5,5
2 Входное напряжение низкого уровня, В	U_{IL}	–0,3	0,7
3 Входное напряжение высокого уровня, В	U_{IH}	2,4	U_{CC}
4 Выходной ток низкого уровня, мА	I_{OL}	-	2
5 Выходной ток высокого уровня, мА	I_{OH}	–0,3	-
6 Напряжение на выходе с 3-м состоянием в состоянии "выключено", В	U_{OZ}	–0,3	U_{CC}
7 Тактовая частота для ИМС 1867ВЦ2Т, МГц	f_{CI}		40
8 Тактовая частота для ИМС 1867ВЦ2АТ, МГц	f_{CI}		57
9 Емкость нагрузки, пФ	C_L	-	50

2.4 Корпус микросхем

Процессор 1867ВЦ2Т (1867ВЦ2АТ) собирается в 132-выводном планарном метало-керамическом корпусе 4229.132-3 ("Монополия-132"). На рисунке 2.2 показан внешний вид корпуса 4229.132-3.

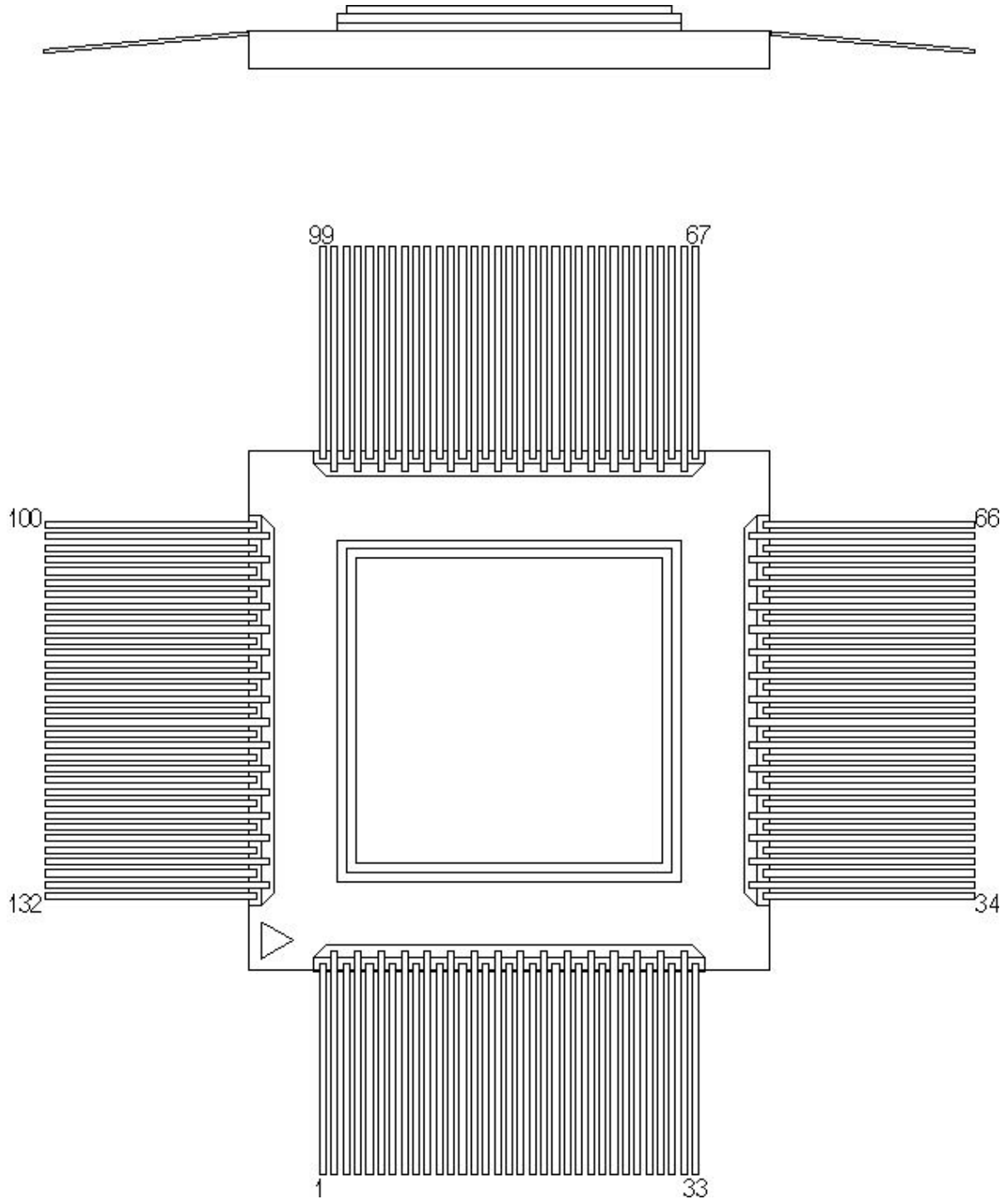


Рисунок 2.2 – Внешний вид корпуса 4229.132-3

В таблице 2.5 перечисляются сигналы, соответствующие выводы корпуса, функциональное назначение и тип вывода, т.е. вход (I), выход (O), активное (S) или высокоомное (Z) состояние. Сигналы в таблице 2.5 сгруппированы в соответствии с их функциональным назначением.

Таблица 2.5. Описание сигналов 1867ВЦ2Т, 1867ВЦ2АТ

Обозначение вывода	Тип вывода ИС	Номер вывода корпуса	Функциональное назначение вывода
1	2	3	4
Выводы шин адреса и данных			
A15	I/O/Z	92	Вывод 15 разряда шины адреса
A14	I/O/Z	91	Вывод 14 разряда шины адреса
A13	I/O/Z	90	Вывод 13 разряда шины адреса
A12	I/O/Z	89	Вывод 12 разряда шины адреса
A11	I/O/Z	88	Вывод 11 разряда шины адреса
A10	I/O/Z	87	Вывод 10 разряда шины адреса
A09	I/O/Z	80	Вывод 09 разряда шины адреса
A08	I/O/Z	79	Вывод 08 разряда шины адреса
A07	I/O/Z	78	Вывод 07 разряда шины адреса
A06	I/O/Z	77	Вывод 06 разряда шины адреса
A05	I/O/Z	76	Вывод 05 разряда шины адреса
A04	I/O/Z	75	Вывод 04 разряда шины адреса
A03	I/O/Z	74	Вывод 03 разряда шины адреса
A02	I/O/Z	73	Вывод 02 разряда шины адреса
A01	I/O/Z	72	Вывод 01 разряда шины адреса
A00	I/O/Z	71	Вывод 00 разряда шины адреса
D15	I/O/Z	23	Вывод 15 разряда шины данных
D14	I/O/Z	24	Вывод 14 разряда шины данных
D13	I/O/Z	25	Вывод 13 разряда шины данных
D12	I/O/Z	26	Вывод 12 разряда шины данных
D11	I/O/Z	27	Вывод 11 разряда шины данных
D10	I/O/Z	28	Вывод 10 разряда шины данных
D09	I/O/Z	29	Вывод 09 разряда шины данных
D08	I/O/Z	30	Вывод 08 разряда шины данных
D07	I/O/Z	38	Вывод 07 разряда шины данных
D06	I/O/Z	39	Вывод 06 разряда шины данных
D05	I/O/Z	40	Вывод 05 разряда шины данных
D04	I/O/Z	41	Вывод 04 разряда шины данных
D03	I/O/Z	42	Вывод 03 разряда шины данных
D02	I/O/Z	43	Вывод 02 разряда шины данных

Продолжение таблицы 2.5

1	2	3	4
D01	I/O/Z	44	Вывод 01 разряда шины данных
D00	I/O/Z	45	Вывод 00 разряда шины данных
Выводы сигналов инициализации, прерываний и сброса			
$\overline{\text{IACK}}$	O/Z	127	Выход сигнала подтверждения прерывания
$\overline{\text{INT1}}$	I	53	Вход внешнего прерывания 1
$\overline{\text{INT2}}$	I	54	Вход внешнего прерывания 2
$\overline{\text{INT3}}$	I	55	Вход внешнего прерывания 3
$\overline{\text{INT4}}$	I	56	Вход внешнего прерывания 4
$\overline{\text{NMI}}$	I	57	Вход немаскируемого прерывания
$\overline{\text{RS}}$	I	12	Вход сброса
MP / $\overline{\text{MC}}$	I	22	Вход выбора режима микропроцессор/микрокомпьютер
Выводы сигналов организации мультипроцессорных систем			
$\overline{\text{BIO}}$	I	15	Вход управления инструкцией перехода
XF	O/Z	125	Выход сигнала (флага) общего назначения
$\overline{\text{HOLD}}$	I	14	Вход активизации состояния удержания
$\overline{\text{HOLDA}}$	O/Z	124	Выход сигнала подтверждения удержания состояния
$\overline{\text{BR}}$	I/O/Z	109	Вывод сигнала запроса шины данных, вход/выход
$\overline{\text{IAQ}}$	O/Z	18	Вывод строба выполнения инструкции
Выводы сигналов управления памятью			
$\overline{\text{DS}}$	O/Z	104	Вывод сигнала выбора пространства данных
$\overline{\text{IS}}$	O/Z	105	Вывод сигнала выбора устройств ввода/вывода
$\overline{\text{PS}}$	O/Z	106	Вывод сигнала выбора пространства программ
$\overline{\text{STRB}}$	I/O/Z	108	Вывод сигнала стробирования, вход/выход
READY	I	13	Вывод сигнала готовности внешнего устройства

Продолжение таблицы 2.5

1	2	3	4
$R/\overline{W}$	I/O/Z	107	Вывод сигнала чтения/записи
$\overline{RD}$	O/Z	95	Вывод разрешения чтения
$\overline{WE}$	O/Z	97	Вывод разрешения записи
Выводы сигналов осциллятора/таймера			
CLKOUT1	O/Z	126	Вывод генератора машинного цикла
CLKMD1	I	119	Вывод выбора режима тактирования 1
CLKMD2	I	86	Вывод выбора режима тактирования 2
CLKIN2	I	110	Вывод подключения внешнего генератора
X2/CLKIN	I	111	Вывод подключения кварцевого резонатора / вход внешней синхронизации
X1	O	112	Вывод подключения кварцевого резонатора
TOUT	O/Z	7	Выход сигнала таймера
Выводы сигналов последовательного порта			
CLKR	I	61	Вход синхросигнала приема по последовательного порта
CLKX	I/O/Z	9	Вывод синхросигнала передачи по последовательному порту
DR	I	58	Вход данных последовательного порта
DX	O/Z	122	Выход данных последовательного порта
FSR	I	60	Вывод старт-импульса приёма по последовательному порту
FSX	I/O/Z	120	Вывод старт-импульса передачи по последовательному порту
Выводы сигналов последовательного порта с временным разделением			
TCLKR	I	11	Вход синхросигнала приема по последовательному порту с разделением по времени
TCLKX	I/O/Z	8	Вывод синхросигнала передачи по последовательному порту с разделением по времени
TDR	I	59	Вход приема данных последовательного порта с разделением времени

Продолжение таблицы 2.5

1	2	3	4
TDX	O/Z	123	Выход передачи данных последовательного порта с разделением времени
TFSR/TADD	I/O/Z	10	Вывод кадрового импульса синхронизации приема данных / адрес порта в режиме разделения времени
TFSX/TFRM	I/O/Z	121	Вывод старт-импульса передачи по последовательному порту с разделением по времени / фрейм данных
Выводы сигналов JTAG порта			
TCK	I	50	Вход сигнала тактирования JTAG-тестирования
TDI	I	83	Вход входных данных JTAG-тестирования
TDO	O/Z	115	Выход выходных данных JTAG-тестирования
TMS	I	46	Вход сигнала выбора JTAG-тестирования
$\overline{\text{TRST}}$	I	19	Вход сигнала включения JTAG-тестирования
EMU0	I/O/Z	3	Вывод нулевого разряда эмулятора
EMU1/$\overline{\text{OFF}}$	I/O/Z	4	Вывод первого разряда эмулятора / перевод Z-выводов в высокоимпедансное состояние
Выводы «земли» и питания			
GND	S	5, 6	Шина «земли» выходных буферов
U_{CC2}	S	16, 17	Шина питания выходных буферов
GND	S	20, 21	Общая шина ядра и входных буферов
U_{CC2}	S	31, 33	Питание выходных буферов шины данных
GND	S	36, 37	Общая шина выходных буферов
U_{CC2}	S	47, 48	Питание выходных буферов шины данных
GND	S	51, 52	Общая шина выходных буферов данных
U_{CC2}	S	62, 64	Питание выходных буферов адресной шины

Продолжение таблицы 2.5

1	2	3	4
GND	S	69, 70	Общая шина выходных буферов адреса
U_{CC1}	S	81, 82	Шина питания ядра и входных буферов
GND	S	84, 85	Общая шина выходных буферов адреса
U_{CC2}	S	93, 94	Питание выходных буферов адресной шины
GND	S	102, 103	Общая шина выходных буферов управления
U_{CC2}	S	113, 114	Шина питания выходных буферов управления
GND	S	116, 118	Общая шина ядра и входных буферов
U_{CC1}	S	128, 130	Шина питания ядра и входных буферов
NC		1, 2, 32, 34, 35, 49, 63, 65-68, 96, 98 - 101, 117, 129, 131, 132	Не задействованы

3 Общая характеристика ИМС

Данный процессор имеет раздельные шины для памяти программ и памяти данных, дополнительные внутрикристальные периферийные устройства, большой размер внутрикристальной памяти и более высокую специализацию системы команд. Это обеспечивает операционную гибкость и высокую производительность ПЦОС. Процессор 1867ВЦ2Т (1867ВЦ2АТ) спроектирован таким образом, чтобы выполнять более 50 MIPS (миллионов операций в секунду). Также реализована совместимость программ процессора 1867ВЦ2Т (1867ВЦ2АТ) с М1867ВМ1 и 1867ВМ2.

Высокая эффективность данного процессора обеспечивает лучшие, более многосторонние подходы к традиционным вопросам обработки сигналов таким, как вокодирование и фильтрация. Процессор 1867ВЦ2Т (1867ВЦ2АТ) поддерживает сложные программы, которые часто требуют одновременного выполнения нескольких операций. Применение возможно в следующих областях обработки сигналов:

Телекоммуникационные системы:

- Эхокомпенсация
- ADPCM транскодеры
- DTMF кодирование/декодирование
- Обработка изображений
- 1200...19200 bps модемы
- Сотовая телефония
- Цифровая речевая интерполяция
- X.25 пакет коммутации
- Видеоконференции

Речевые системы:

- Речевое кодирование
- Распознавание речи
- Верификация речи
- Преобразование текста в речь

Системы ЦОС:

- Цифровая фильтрация
- Свертка, корреляция
- Преобразование Гильберта
- Быстрое преобразование Фурье
- Адаптивная фильтрация

Процессор 1867ВЦ2Т(1867ВЦ2АТ) совместим по системе команд с М1867ВМ1 и 1867ВМ2 и наследует их общие архитектурные особенности. Усовершенствования включают 32-разрядный аккумуляторный буфер, дополнительные возможности масштабирования и новые команды для поддержки дополнительных аппаратных средств. Новые функции управления включают независимое параллельное логическое устройство (PLU) для выполнения булевых операций. Реализация одноуровневого аппаратного стека для основных регистров процессора (наличие так называемых «теневых» регистров) обеспечивает возможность аппаратного переключения контекста и существенного сокращения программных (а значит временных) затрат при вызовах/возвратах подпрограмм обработки прерываний (ISR). Оптимизация потоков данных была достигнута путем введения новых команд перемещения блоков и команд обслуживания картированных в памяти регистров. 1867ВЦ2Т (1867ВЦ2АТ) имеет 28 регистров ядра (Центрального Процессорного Устройства – ЦПУ), картированных в памяти и 16 параллельных портов ввода/вывода, картированных в памяти.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит 2К х 16-разрядное внутрикристальное, масочно программируемое ПЗУ (только для чтения). Эта память используется в основном для хранения программы начальной загрузки (boot loader), которая обеспечивает несколько различных вариантов закачки пользовательского программного кода в большое внутрикристальное ОЗУ. Если при сбросе на входе $MP/\overline{MC}$ был установлен низкий уровень (режим микрокомпьтер), то процессор начнет исполнять программный код, содержащийся во внутрикристальном ПЗУ. Как только программа пользователя будет загружена во внутрикристальное ОЗУ, внутрикристальное ПЗУ может быть удалено из пространства памяти программ путем программной установки бита $MP/\overline{MC}$ в единицу (регистр PMST). Если при сбросе на входе $MP/\overline{MC}$ установить высокий уровень (режим микропроцессор), будет выполняться программа, расположенная во внешней памяти. Отметим, что изменение сигнала на входе $MP/\overline{MC}$ актуально только во время сброса, а в дальнейшем не оказывает никакого влияния на соответствущий бит.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет внутрикристальное 1056-словное ОЗУ двойного доступа (ДОЗУ). ДОЗУ разделено на три независимо выбираемых блока памяти: 512-слов данных или программ В0, 512-слов данных В1 и 32-слова данных В2. Блоки В1 и В2 всегда сконфигурированы в память данных; а блок В0 может быть программно сконфигурирован как в память данных, так и в программ. ДОЗУ может быть сконфигурировано одним из двух способов:

- все 1056 слов сконфигурированы в память данных;

– 544 слов сконфигурированы в память данных и 512 слов сконфигурированы в память программ.

ДОЗУ способно реально увеличить скорость работы процессора. Благодаря раздельным шинам адреса чтения и адреса записи и раздельным шинам данных для чтения и для записи, 1867ВЦ2Т (1867ВЦ2АТ) способен считывать из, и писать в ДОЗУ в одном и том же машинном цикле. Необходимо отметить, что процессор работает в четырёх-уровневом конвейере. При этом чтение данные происходит в третьей, а запись данных в четвёртой стадии конвейера. Поэтому вторая инструкция чтения данных выполняется одновременно с первой инструкцией записи данных.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит 9К × 16-битное внутрикристальное оперативное запоминающее устройство одинарного доступа (ОЗУ). Выборка программного кода из области внешней памяти требует в лучшем случае одного дополнительного машинного цикла. Вместо этого код может быть загружен в большое внутрикристальное ОЗУ и затем выполнен на полной скорости.

ОЗУ может быть программно сконфигурировано одним из трёх способов:

– всё ОЗУ сконфигурировано в память данных;

– всё ОЗУ сконфигурировано в память программ;

– ОЗУ сконфигурировано одновременно в память данных и в память программ.

ОЗУ разбито на блоки: четыре по 2К-слов и один 1К-слов. ЦПУ поддерживает параллельный доступ к этим блокам ОЗУ. Если инструкция чтения и/или записи обращается одновременно к разным блокам ОЗУ, логика доступа распределяет запросы, так чтобы обеспечить обмен в течение одного машинного цикла. В случае одновременного обращения в один и тот же блок требуется дополнительный машинный цикл.

ОЗУ поддерживает более гибкую адресацию обращения, чем ДОЗУ поскольку может быть сконфигурировано и в пространство памяти данных и в пространство памяти программ одновременно. Но при этом выборка инструкции и данных для нее будет осуществляться за два машинных цикла.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет режим защиты содержимого внутрикристальной памяти программ. Если бит защиты установлен, никакой внешней инструкцией нельзя обратиться к пространству внутрикристальной памяти программ.

Логика программно-задаваемых состояний ожидания 1867ВЦ2Т (1867ВЦ2АТ) позволяет совмещать процессор с внешними устройствами (память, ввод/вывод),

работающими с разной скоростью. Внешнее адресное пространство программ, данных, и ввода/вывода разбивается на диапазоны с индивидуально программируемым для каждого диапазона количеством циклов (0, 1, 2, 3 или 7) ожидания доступа. Таким образом, достигается минимизация внешних аппаратных затрат при построения интерфейса с памятью и портами ввода/вывода.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит два быстродействующих последовательных порта. Один из двух портов является синхронным, дуплексным последовательным портом. Его передатчик и приемник дважды буферизируются и индивидуально управляются маскируемыми внешними сигналами прерывания. Данные оформляются или как байты, или как слова. Второй порт является дуплексным последовательным портом, который может конфигурироваться или для синхронных операций, или для операций множественного доступа с квантованием по времени (TDM). Последовательный порт TDM часто используется в мультипроцессорных системах.

В процессоре 1867ВЦ2Т (1867ВЦ2АТ) предусмотрен 16-битный аппаратный таймер с 4-разрядным предварительным делителем. Это программируемый генератор импульсов с частотой в диапазоне от 1/2 до 1/32 длительности машинного цикла (CLKOUT1), в зависимости от содержтмого регистра делителя. Таймер может быть остановлен, перезапущен, сброшен или отключен установкой соответствующих разрядов его управляющего регистра.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет четыре линии внешних прерываний ($\overline{\text{INT1}} - \overline{\text{INT4}}$) и пять внутренних прерываний: одно от таймера и четыре от последовательных портов, маскируемые пользователем. Когда выполняется прерывание текущей программы, содержимое программного счётчика сохраняется в 8-ми уровневом аппаратном стеке, и содержимое одиннадцати специальных регистров ЦПУ автоматически сохраняется в одноуровневом аппаратном стеке. Когда выполняется инструкция возврата из прерывания, содержимое регистров ЦПУ автоматически восстанавливается.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) адресует до 64К пространства параллельных портов ввода/вывода; шестнадцать из которых картированы в памяти данных. Эти порты могут быть адресованы командами IN или OUT. Картированные в памяти порты доступны для любой инструкции записи или чтения в/из памяти. Низкий уровень на выводе $\overline{\text{IS}}$ указывает на операцию чтения/записи через порт ввода/вывод. Процессор 1867ВЦ2Т (1867ВЦ2АТ) может быть легко сопряжен с внешними устройствами через порты

ввода/вывода с минимальными внешними аппаратными затратами для организации логики декодирования адреса.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит интерфейс тестового последовательного доступа в соответствии со стандартом IEEE 1149.1 (JTAG). Этот интерфейс предусматривает работу с логикой граничного сканирования выводов (Boundary Scan Register), и используется для эмуляции и тестирования. Эта логика обеспечивает последовательное граничное сканирование (и в режиме считывания и в режиме подачи информации на контактные площадки процессора) как его внутренних устройств, так и его внешнего интерфейса.

Внутрикристальный блок анализа связан с программным отладчиком, способным выполнять отладку и выполнять оценку функций в заданной системе. Блок анализа реализует следующие возможности:

- Установка точки останова. Точки останова могут быть назначены для следующих событий:
  - выборка/чтение/запись кодов программ;
  - активизация выводов EMU0/1;
  - чтение/запись данных;
  - события ЦПУ (вызовы, возвраты, прерывания, переходы, тактирование конвейера).
  -
- Для функционального анализа определяются следующие состояния (события):
  - тактирование ЦПУ;
  - прохождение инструкций по конвейеру;
  - выборка инструкций;
  - вызовы, возвраты, прерывания, переходы;
  - выборка/чтение/запись кодов программ;
  - чтение/запись данных.
- Мониторинг последовательности выполнения команд с помощью буфера трассировки программных адресов.

4 Описание устройства

4.1 Внутренняя архитектура

Архитектура процессора ЦОС 1867ВЦ2Т (1867ВЦ2АТ) базируется на следующих основных элементах:

- шинная структура;
- центральное процессорное устройство (ЦПУ);
- внутрикристальная память;
- внутренние периферийные устройства.

Эта глава описывает архитектуру и работу ЦПУ 1867ВЦ2Т (1867ВЦ2АТ) с оперативной памятью.

4.1.1 Архитектурный обзор

Процессор 1867ВЦ2Т (1867ВЦ2АТ), подобно 1867ВМ2, спроектирован в соответствии с архитектурой Гарвардского типа, которая позволяет осуществлять одновременный доступ к программной памяти и памяти данных, обеспечивая высокую степень параллелизма. Например, пока данные проходят через умножитель, результат предыдущего умножения, может быть прибавлен или вычтен из содержимого аккумулятора и, в течении этого же времени, происходит генерация очередных адресов.

1867ВЦ2Т (1867ВЦ2АТ) реализует арифметику в двоичной системе, используя 32-разрядное ALU и аккумулятор. ALU является арифметическим устройством общего назначения, которое работает, используя 16-разрядные операнды, взятые из памяти данных, из программной памяти (непосредственные операнды), или используя 32-разрядное произведение с выхода умножителя. В дополнение к арифметическим операциям ALU может выполнять Булевы операции. Аккумулятор хранит выходные данные из ALU и обеспечивает подачу данных на второй вход ALU. Аккумулятор имеет длину в 32 разряда и разделяется на старшее (с 31 по 16 разряды) и младшее(с 15 по 0 разряды) слова. Предусмотрены команды для сохранения в памяти обоих слов ккумулятор. Для быстрого временного сохранения аккумулятора имеется 32-разрядный аккумуляторный буфер. В дополнение к главному ALU имеется параллельное логическое устройство (PLU), которое выполняет логические операции над данными, не затрагивая содержимого аккумулятора. PLU обеспечивает возможности управления битами, необходимые высокоскоростному контроллеру и упрощает установку, сброс и тестирование битов, необходимых для операций с регистрами состояния и управления.

Умножитель выполняет 16 x 16 -разрядное умножение в двоичной системе с 32-разрядным результатом за один машинный цикл. Он состоит из трех элементов: массива умножителя, PREG (регистра результата) и TREG0 (регистра сомножителя). 16 -разрядный TREG0 временно хранит множитель, PREG хранит 32 -разрядный результат. Множимое на второй вход умножителя поступает: из памяти данных, из памяти программ (когда используются команды MAC/MACD/MADS/MADD или команды умножения с непосредственным операндом MPY #). Быстрый внутрикристальный умножитель позволяет устройству эффективно выполнять основополагающие операции ЦОС такие, как свертка, корреляция и фильтрация.

Сдвиговый регистр масштабирования 1867ВЦ2Т (1867ВЦ2АТ) имеет 16-разрядный вход, соединенный с шиной данных и 32-разрядный выход, подключенный к АЛУ. Сдвиговый регистр производит сдвиг входных данных влево от 0 до 16 разрядов. Коэффициент задается либо непосредственно в команде или определяется содержимым регистра счетчика сдвига (TREG1). Выходы младших разрядов (LSB) заполняются нулями, в то время, как старшие разряды могут быть либо заполнены нулями, либо расширены значением знакового бита в зависимости от состояния разряда режима расширения знака (SXM) в регистре состояния ST1. Дополнительные возможности сдвига позволяют процессору выполнять операции цифрового масштабирования, выделения битов, расширенной арфметики и ают предотвращают переполнение.

Восемь уровней аппаратного стека предусмотрены для сохранения содержимого программного счетчика во время прерываний и вызовов подпрограмм. При переходе к подпрограмме обработки аппаратного прерывания, стратегические регистры (ACC, ACCB, ARCR, INDX, PMST, PREG, ST0, ST1, TREG) помещаются в одноуровневый стек и могут быть извлечены из него при возврате из прерывания.

4.1.2 Функциональная блок-диаграмма

Функциональная блок-диаграмма, представленная на рисунке 4.1, содержит основные блоки и показывает маршруты данных в процессорах 1867ВЦ2Т, 1867ВЦ2АТ. Архитектура процессоров включает в себя две основные шины: шину программ, передающую командные коды и непосредственные операнды из программной памяти и шину данных, соединяющую различные устройства, такие как арифметико-логическое устройство (ALU) и файл вспомогательных регистров с памятью данных. Вместе шины программ и данных могут одновременно подавать сомножители из внутрикристальной памяти данных и внутренней или внешней памяти программ на оба входа умножителя для обеспечения полнопоточного одноциклового умножения или умножения/накопления.

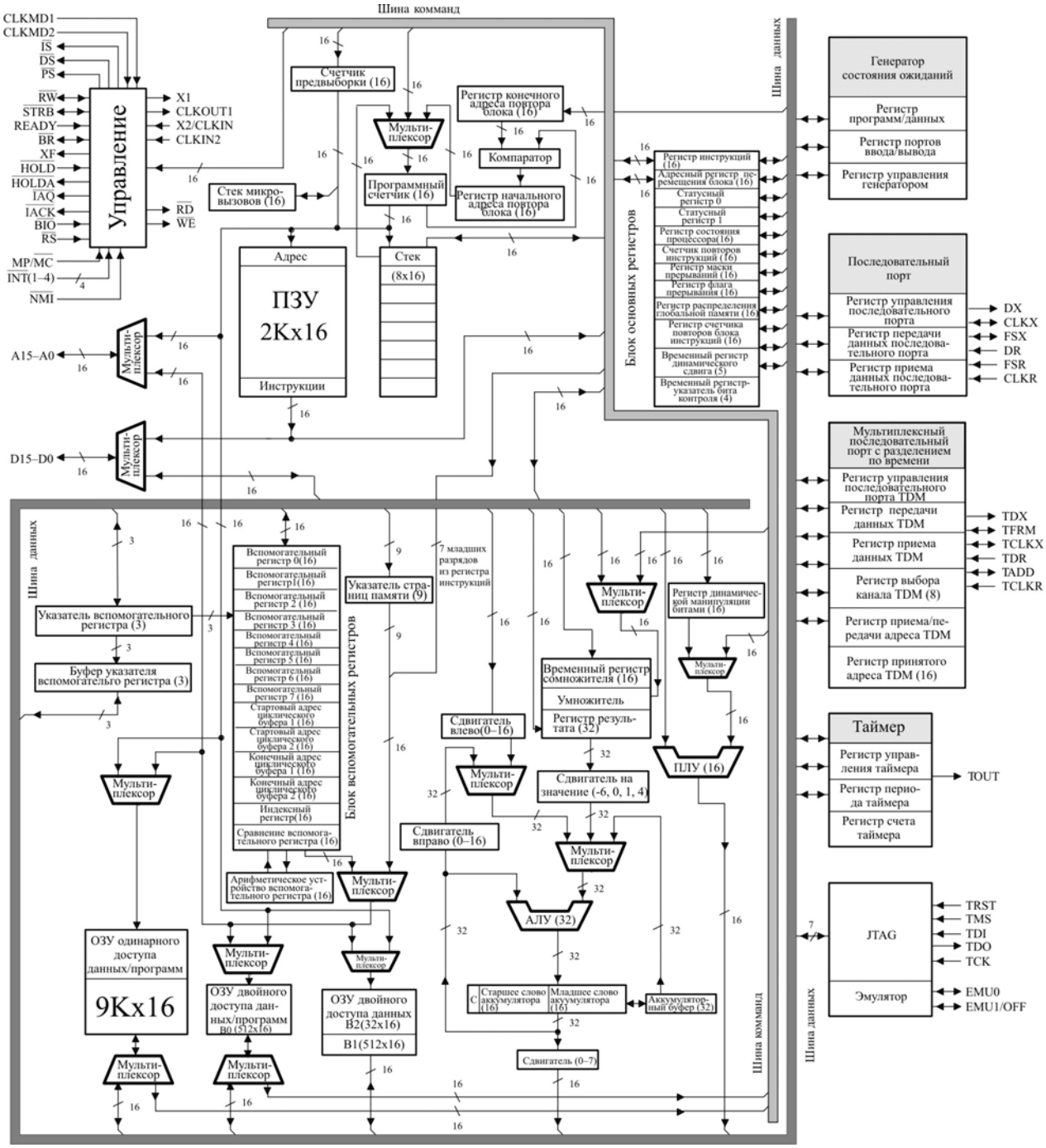


Рисунок 4.1 - Блок-диаграмма внутренних аппаратных средств 1867ВЦ2Т, 1867ВЦ2АТ

4.1.3 Краткий обзор внутренних аппаратных средств

Внутренние аппаратные средства 1867ВЦ2Т (1867ВЦ2АТ) позволяют аппаратно выпонять функции, которые в других процессорох обычно реализуются программно или в микрокодах. Например, устройство содержит аппаратные средства для одноциклового 16 × 16 - разрядного умножения, сдвига данных и управления адресами.

Таблица 4.1 представляет краткий обзор внутренних аппаратных средств 1867ВЦ2Т, 1867ВЦ2АТ. Эта сводная таблица включает элементы внутренней архитектуры, регистры и шины в алфавитном порядке. Все символы, использованные в таблице, соответствуют символам, использованным на рисунке 4.1, последующим блок-диаграммам в этом разделе и тексту всего документа.

Таблица 4.1 - Краткий обзор внутренних аппаратных средств 1867ВЦ2Т, 1867ВЦ2АТ

Устройство	**Аббревиатура**	**Описание**
1	2	3
Аккумулятор	ACC (32) ACCH(16) ACCL (16)	32-разрядный аккумулятор. Состоит из двух частей: ACCH (старшее слово) и ACCL (младшее слово аккумулятора). Используется для хранения данных ALU.
Буфер аккумулятора	ACCB(32)	Регистр используется для временного хранения 32-разрядного содержимого аккумулятора. Этот регистр непосредственно соединен с ALU и может совместно с ACC быть использован в вычислениях с двойной точностью.
Арифметико-логическое устройство	ALU	32-разрядное арифметико-логическое устройство для операций в двоичных кодах, имеющее два 32-разрядных входа и один 32-разрядный выход (на аккумулятор)
Арифметическое устройство вспомогательных регистров	ARAU	Беззнаковое 16-разрядное арифметическое устройство, применяемое для вычисления косвенного адреса, использует в качестве входов вспомогательные, индексные регистры и регистры сравнения
Региср сравнения	ARCR(16)	16-разрядный регистр, используемый в качестве граничного компаратора при косвенной адресации
Файл вспомогательных регистров	AUXREGS	Регистровый файл, содержащий восемь 16-разрядных вспомогательных регистров (AR0-AR7), используемых для хранения косвенного адреса данных, временного хранения или арифметической обработки чисел посредством ARAU
Буфер указателя вспомогательного регистра	ARB(3)	3-разрядный регистр, который содержит предыдущее значение, содержащееся в ARP. Эти разряды хранятся в ST1
Указатель вспомогательного регистра	ARP(3)	3-разрядный регистр, используемый в качестве указателя выбранного в настоящий момент вспомогательного регистра. Эти разряды хранятся в ST0
Регистр адреса перемещения блока	BMAR(16)	16-разрядный регистр, который содержит значение адреса используемое при перемещении блока или в операциях умножения с накоплением

Продолжение таблицы 4.1

1	2	3
Флаг повтора блока	BRAF(1)	1-разрядный флаг, указывающий, что в настоящий момент активирован повтор блока. Устанавливается, при выполнении команды RPTB и сбрасывается, когда BRCR становится равным нулю. Расположен в регистре PMST
Регистр конечного адреса повтора блока	PAER(16)	16-разрядный регистр, содержащий конечный адрес сегмента повторяемого кода
Регистр стартового адреса повтора блока	PASR(16)	16-разрядный регистр, содержащий стартовый адрес сегмента повторяемого кода
Регистр счетчика повтора блока	BRCR(16)	16-разрядный регистр счетчика количества повторов программного сегмента (при организации аппаратных циклов)
Модуль межшинного интерфейса	BIM	Буферизированный интерфейс, используемый для обмена между шинами данных и программ
Запрос шины	$\overline{\text{BR}}$	Этот сигнал используется для арбитрирования доступа к совместной внешней (глобальной) памяти в мультипроцессорных системах и при организации ПДП
Перенос	C	Этот разряд хранит перенос АЛУ. Расположен в ST1
Центральное арифметико-логическое устройство	CALU	Тракт обработки данных: АЛУ, умножитель, аккумулятор и масштабирующие сдвиговые регистры
Регистр управления циклическим буфером	CBCR(8)	8-разрядный регистр, используемый, для включения (выключения) циклического буфера и указания, какие вспомогательные регистры связаны с циклическими буферами
Регистр конечного адреса циклических буферов (1,2)	CBER(16) CBER1(16) CBER2(16)	Два 16-разрядных регистра, указывающие конечные адреса циклического буфера. CBER1 и CBER2 связаны с циклическими буферами один и два соответственно
Сравнение адреса программы	COMPARE	Логика сравнения текущего значения PC со значением в PAER, когда активирован BRAF (при аппаратных циклах). В случае равенства PC загружается значением PASR
Конфигурация памяти	CNF	Конфигурирование внутрикристального ОЗУ двойного доступа (в пространство памяти программы или данных). Разряд CNF располагается в ST1
Шина данных	DATA BUS	16-разрядная шина, используемая для передачи данных
Память данных	DATA MEMORY	Общее пространство памяти данных (внутрикристальной и внешней)
Адресная шина памяти данных	DATA ADDRESS	16-разрядная шина, которая осуществляет передачу адресов для памяти данных
Регистр непосредственного адреса памяти данных	DMA(7)	7-разрядный регистр, содержащий непосредственный адрес в пределах страницы данных в 128 слов
Указатель страниц памяти данных	DP(9)	9-разрядный регистр, содержащий адрес текущей страницы. Общее адресуемое пространство памяти данных делится на 512 страниц по 128 слов в каждой (некоторые ячейки зарезервированы). Полный 16-ти разрядный адрес данных образуется при конкатенации значений DP(9) – старшие и DMA(7) младшие разряды
Бит конфигурирования ОЗУ в память программ	RAM(1)	Высокий уровень этого бита конфигурирует ОЗУ однократного доступа в пространство памяти программ См. также OVLY бит. Расположен в PMST
Шина прямого адреса памяти данных	DRB(16)	16-разрядная шина прямого адреса памяти данных — результат конкатенации DP(9) и семи младших разрядов для инструкций с прямой адресацией DMA(7)
Регистр динамической манипуляции битами	DBMR(16)	16-разрядный регистр, используемый в качестве программируемой маски для логических операций в PLU (за исключением инструкций PLU с длинным непосредственным операндом)

Продолжение таблицы 4.1

1	2	3
Регистр динамического указателя битов	TREG2(4)	4-разрядный регистр, который содержит 16-тиричное значение номера тестируемого бита для команды BITT
Регистр управления входным сдвигателем ALU	TREG1(5)	5-разрядный регистр коэффициента сдвига для предварительного масштабирования данных на входе ALU
Бит внешнего флага	XF(1)	Этот разряд определяет уровень на выходе внешнего флага XF. Расположен в ST1
Регистр распределения глобальной памяти	GREG(8)	8-разрядный регистр определяющий соотношение размеров локального и глобального пространства памяти данных
Режим HOLD	HM(1)	При низком уровне этого бита в режиме удержания, ЦПУ продолжает выполнение операций с внутрикристальной памятью (до первого обращения вовне). Расположен в ST1
Индексный регистр	INDX(16)	16-разрядный регистр, используемый в разновидностях косвенной адресации с шагом модификации, отличным от ±1
Бит совместимости вспомогательных регистров	NDX(1)	Режим совместимости с объектными кодами 1867ВМ2, в котором функции регистров INDX и ARCR выполняет AR0. При 1 INDX и ARCR являются самостоятельными регистрами и не связаны с AR0. Расположен в PMST
Регистр флагов прерываний	IFR(16)	16-разрядный флаговый регистр. Низкий уровень в соответ-свующем разряде(ах) индицирует активное прерывание(я). IFR является картированным в памяти регистром
Бит режима прерываний	INTM(1)	Высокий уровень запрещает все маскируемые прерывания. Расположен в ST0
Регистр маски прерывания	IMR(16)	16-разрядный регистр, используемый для маскирования прерываний
Указатель станицы векторов прерываний	IPTR(5)	Пять разрядов, определяющих в какой из 32 2К-словных страниц прграммной памяти расположены вектора прерываний. При сбросе обнуляются (соответствует нулевой странице), содержатся в PMST
Режим микропроцессор/ микрокомпьютер	MP/$\overline{\text{MC}}$	Равный 0 конфигурирует внутрикристальное ПЗУ в память программ. Расположен в PMST
Умножитель	MULTIPLIER	16 × 16-разрядный параллельный умножитель
Флаг переполнения	OV(1)	Этот разряд указывает на переполнение при арифметических операциях в ALU. Расположен в ST0
Режим обработки переполнения	OVM(1)	При высоком уровне переполнение приводит к записи в ACC (в зависимости от знака результата) или максимального положительного (00 7FFF FFFFh) или отрицательного (FF 8000 00000h) числа. Иначе результат не изменяется, старшие биты, превышающие длину ACC теряются. Расположен в ST0
Бит конфигурирования ОЗУ в память данных	OVLY(1)	Высокий уровень этого бита конфигурирует ОЗУ однократного доступа в пространство памяти данных. См. также RAM бит. Расположен в PMST
Параллельное логическое устройство	PLU	16-разрядное логическое устройство, выполняющее логические операции независимо от ALU. Маской логической операции служит или содержимое DBMR или длинный непосредственный операнд
Регистр результата умножения	PREG(32)	32-разрядный регистр результата, используемый для хранения результата умножения. Старшие и младшие 16-ти разрядные слова этого регистра могут быть сохранены в память индивидуально
Шина программ	PROG BUS	16-разрядная шина, используемая для предачи инструкций
Счетчик программного адреса	PC(16)	16-разрядный счетчик памяти программ
Память программ	PROGRAM MEMORY	Общее пространство памяти программ (внутрикристальной и внешней)

Продолжение таблицы 4.1

1	2	3
Шина программного адреса	PROG ADDRESS	16-разрядная шина адреса программной памяти
Сдвиговый регистр предварительного масштабирования	PRESCALER	0-16 – разрядный регистр сдвига (влево и вправо), используемый для предварительного масштабирования 16-ти разрядных данных, поступающих на 32-ух разрядный вход АЛУ, выравнивания данных при вычислениях с двойной точностью
Сдвиговый регистр постмасштабирования	POST-SCALER	0-7 – разрядный регистр сдвига влево, для масштабирования выходных данных ALU
Сдвиговый регистр результата умножения	P-SCALER	0, 1, 4 – разрядный регистр сдвига влево, отбрасывающий (при необходимости) дополнительные знаковые разряды результата умножения или 6-разрядный регистр сдвига вправо, масштабирующий результат, чтобы избежать переполнения
Режим сдвигового регистра результата	PM(2)	Эти два разряда определяют режим регистра сдвига результата умножения. Расположены в ST1
Счетчик повтора инструкции	RPTC(16)	16-разрядный счетчик, используемый для управления количеством повторов выполнения однократной инструкции
Режим знакового расширения	SXM(1)	Этот разряд управляет формированием результата арифметических операций, который либо будет знаково-расширен, либо нет. Расположен в ST1
Стек	STACK	8×16 - разрядный аппаратный стек, используемый для хранения PC во время прерываний и вызовов. Значения ACCL и памяти данных могут также засылаться в стек и выталкиваться из него
Статусные регистры	ST0, ST1, PMST	Три 16-разрядных регистра состояний, которые содержат биты состояний и управления
Регистр временного хранения множителя	TREG0(16)	16-разрядный регистр, временного хранения операнда для умножителя
Бит совместимости TREG	TRM(1)	Режим совместимости с объектными кодами 1867ВМ2, в котором функции регистров TREG1, TREG2 выполняет TREG0. При 1 TREG1 и TREG2 являются самостоятельными регистрами и не связаны с TREG0. Расположен в PMST
Флаг тест/управление	TC(1)	Этот разряд хранит результаты операций тестирования разрядов ALU и PLU. Состояние TC служит признаком для условных операций. Расположен в ST1

4.1.4 Организация памяти

Этот подраздел описывает использование пространства оперативной памяти процессора 1867ВЦ2Т (1867ВЦ2АТ) и режимы адресации, поддерживаемые оперативной памятью.

4.1.4.1 Регистры, картированные в памяти

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит 28 картированных в памяти регистров ядра. Они перечислены в таблице 4.2. Кроме них в нулевой странице пространства данных предусмотрено еще 64 ячейки для картированных в памяти 16 параллельных портов ввода/вывода и регистров данных и управления периферийных устройств.

Таблица 4.2 – Регистры процессорного ядра, картируемые в памяти

Имя	Адрес		Описание
	1867ВЦ2Т, 1867ВЦ2АТ десятичный	1867ВЦ2Т, 1867ВЦ2АТ 16-ричный	
	0-3	0-3	Зарезервировано
IMR	4	4	Регистр маски прерываний
GREG	5	5	Регистр распределения глобальной памяти
IFR	6	6	Регистр флагов прерываний
PMST	7	7	Регистр состояния и режимов процессора
RPTC	8	8	Регистр счетчика повторов инструкции
BRCR	9	9	Регистр счетчика повторов блока инструкций
PASR	10	A	Регистр стартового адреса повторямого блока
PAER	11	B	Регистр конечного адреса повторямого блока
TREG0	12	C	Регистр временного хранения множителя
TREG1	13	D	Регистр коэффициента сдвига
TREG2	14	E	Регистр номера тестируемого бита
DBMR	15	F	Регистр динамического управления битами
AR0	16	10	Нулевой вспомогательный регистр
AR1	17	11	Первый вспомогательный регистр
AR2	18	12	Второй вспомогательный регистр
AR3	19	13	Третий вспомогательный регистр
AR4	20	14	Четвертый вспомогательный регистр
AR5	21	15	Пятый вспомогательный регистр
AR6	22	16	Шестой вспомогательный регистр
AR7	23	17	Седьмой вспомогательный регистр
INDX	24	18	Индексный регистр
ARCR	25	19	Регистр сравнения вспомогательных регистров
CBSR1	26	1A	Регистр стартового адреса 1-го циклического буфера
CBER1	27	1B	Регистр конечного адреса 1-го циклического буфера
CBSR2	28	1C	Регистр стартового адреса 2-го циклического буфера
CBER2	29	1D	Регистр конечного адреса 2-го циклического буфера
CBCR	30	1E	Регистр управления циклическими буферами
BMAR	31	1F	Регистр адреса перемещения блоков

4.1.4.2 Режимы адресации памяти

Процессор 1867ВЦ2Т (1867ВЦ2АТ) может адресовать 64К слов памяти программы и 96К слов памяти данных. Внутрикристальная память данных отображается в 96К пространстве памяти данных, а внутрикристальная память программы отображается в 64К пространстве памяти программ.

В настоящем разделе рассматриваются основные способы генерации адресов данных и приводятся примеры программного кода с комментариями.

- Прямая адресации с использованием указателя страницы данных (DP).

− Адресация картированных впамяти регистров.

− Косвенная адресация с использованием ARAU и файла вспомогательных регистров.

− Непосредственная адресация с использованием короткого и длинного непосредственного операнда.

− Регистровая адресация (с использованием специализированных картированных в памяти регистров BMAR и DBMR).

В режиме прямой адресации содержимое 9-разрядного указателя страницы памяти данных (DP) определяет одну из 512 128-ми словных страниц. Семь младших разрядов инструкции с прямой адресацией (dma) определяют, адрес слова внутри страницы. Адрес на DRB образуется конкатенацией 9-разрядного DP с 7-разрядным dma. Рисунок 4.2 иллюстрирует режим прямой адресации. В приведенном примере операнд выбирается из пространства памяти данных через шину данных, а адрес представляет собой сконкатенированное значение DP и семи младших битов команды.

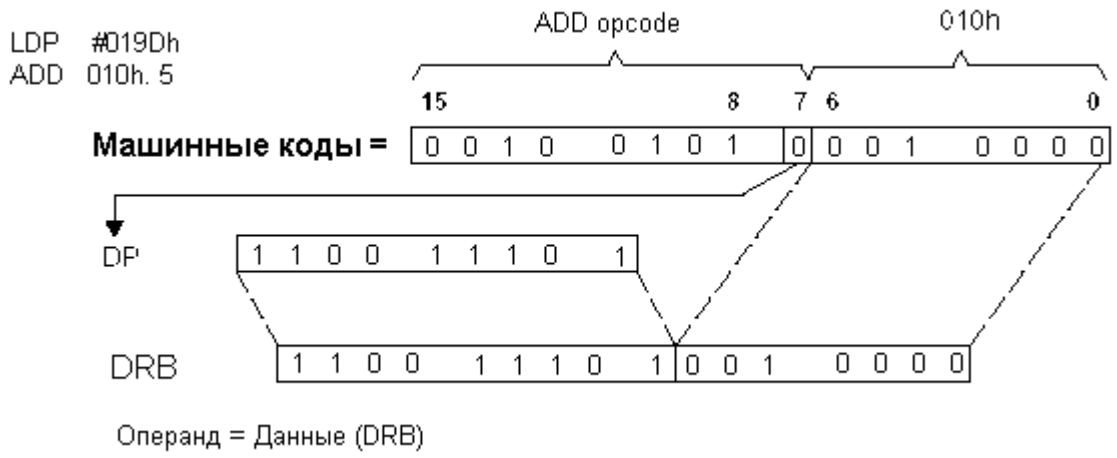
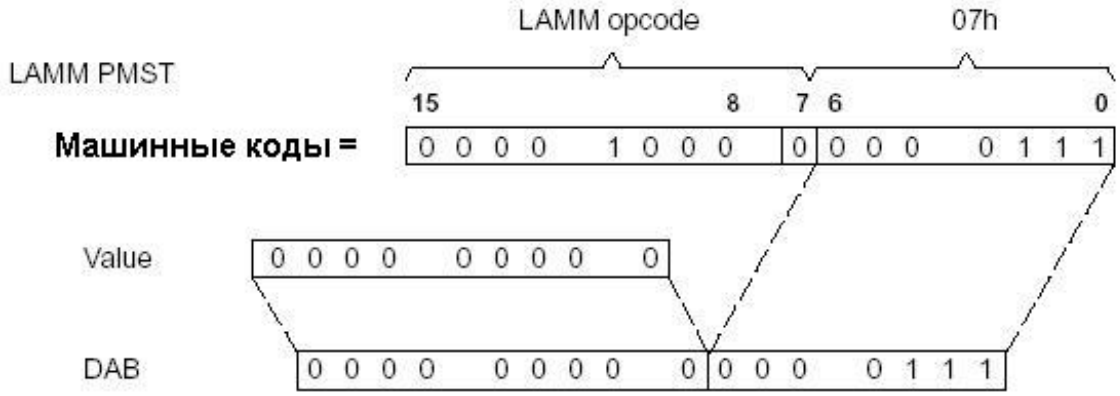


Рисунок 4.2 - Режим прямой адресации

Необходимо отметить, что DP не инициализируется при сбросе, и его состояние после включения питания является неопределенным. Поэтому, для корректной работы с памятью данных, DP необходимо инициализировать программным путем.

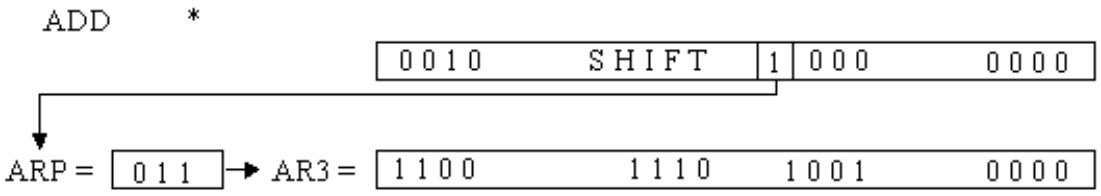
Адресация картированных в памяти регистров реализуется почти также, как и прямая адресации, за исключением того, что 9 старших разрядов адреса всегда принудительно установлены в ноль, независимо от содержимого DP. Это позволяет пользователю напрямую обращаться к картированным в памяти регистрам нулевой

страницы данных без дополнительных затрат времени на изменение DP или вспомогательных регистров. Рисунок 4.3 иллюстрирует режим адресации картированных в памяти регистров.



Операнд = Данные (DRB)
Рисунок 4.3 - Адресация картированных в памяти регистров

В режиме косвенной адресации выбранный (в соответствии с содержимым ARP) в настоящий момент 16-разрядный вспомогательный регистр AR(ARP) адресует память данных через шину файла вспомогательных регистров (AFB). Пока выбранный вспомогательный регистр обеспечивает адрес памяти данных, а над данными выполняется операция ALU, содержимое вспомогательного регистра может быть модифицировано с помощью ARAU. Для примера на рисунке 4.4 показана косвенная адресация с использованием третьего (см. ARP) вспомогательного регистра.

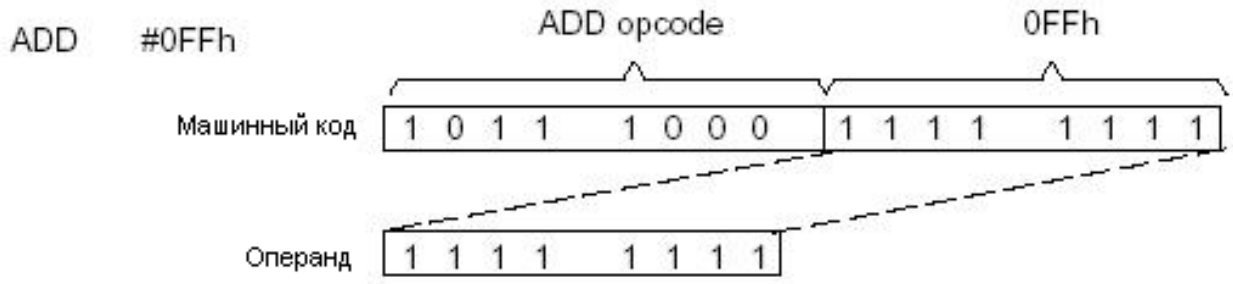


Операнд = Data (AR(ARP))
Рисунок 4.4 - Режим косвенной адресации.

Если косвенная адресация используется в инструкциях, работающих с картированными в памяти регистрами, от AR(ARP) используются только семь младших разрядов, старшие девять в расчет не принимаются. Отметим что признаком косвенной адресации является единица в седьмом бите инструкции. В инструкциях с прямой адресацией седьмой бит равен нулю.

Операнд может являться частью кода инструкции. Короткие непосредственные операнды содержатся в команде, состоящей из единственного слова. Они отличаются по

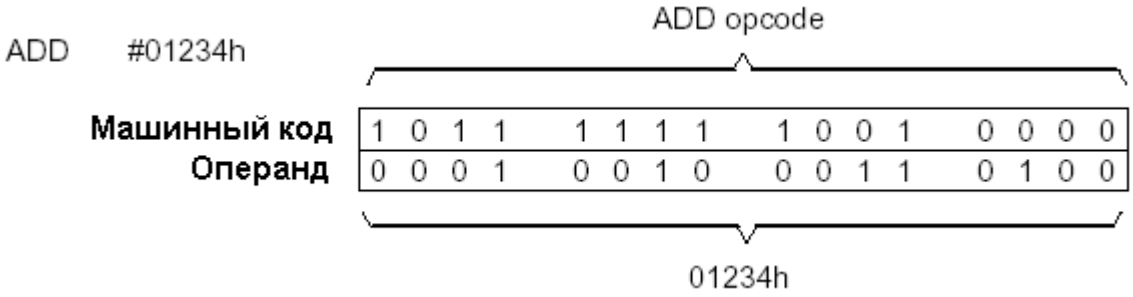
длине от 1 разряда в команде SETC до 13 разрядов в команде MPY #. Рисунок 4.5 иллюстрирует пример режима короткой непосредственной адресации. В этом примере восемь младших разрядов инструкции ADD являются операндом и с помощью ALU добавляются к содержимому ACC.



Операнд = Data (ADD(7-0))

Рисунок 4.5 – Режим короткого непосредственного операнда

Длинные непосредственные операнды следуют вторым словом за 16-ти разрядным кодом операции и также составляют 16 разрядов в длину. Существует две разновидности длинной непосредственной адресации для инструкций с одним и с двумя операндами соответственно. На Рисунке 4.6 приведен пример длинной непосредственной адресации без доступа к памяти программ. В этом примере второе слово двухсловной команды добавляется с помощью ALU к содержимому ACC.



Операнд = Data (второе слово(15-0))

Рисунок 4.6 – Режим длинного непосредственного операнда без доступа к памяти

Длинная непосредственная адресация может применяться одновременно с доступом к памяти данных для выполнения двухоперандной инструкции. Счетчик предвыборки (PFC) сохраняется в стеке микровызовов (MCS), затем длинный непосредственный операнд загружается в PFC. После этого шины программного адреса (PAB) и инструкций (PB) используются для выборки (чтения) или записи операнда. При завершении выполнения инструкции содержимое MCS выталкивается обратно в PFC, программный счетчик (PC) увеличивается на два и исполнение программы продолжается. PFC используется, чтобы обеспечить, при организации повторения инструкции, генерацию адреса с автоинкрементом (увеличением на единицу).

Этот режим иллюстрирует Рисунок 4.7. На этом рисунке адрес источника (OPERAND1) выбирается через PAB, а адрес места назначения (OPERAND2) генерируется в режиме прямой адресации.

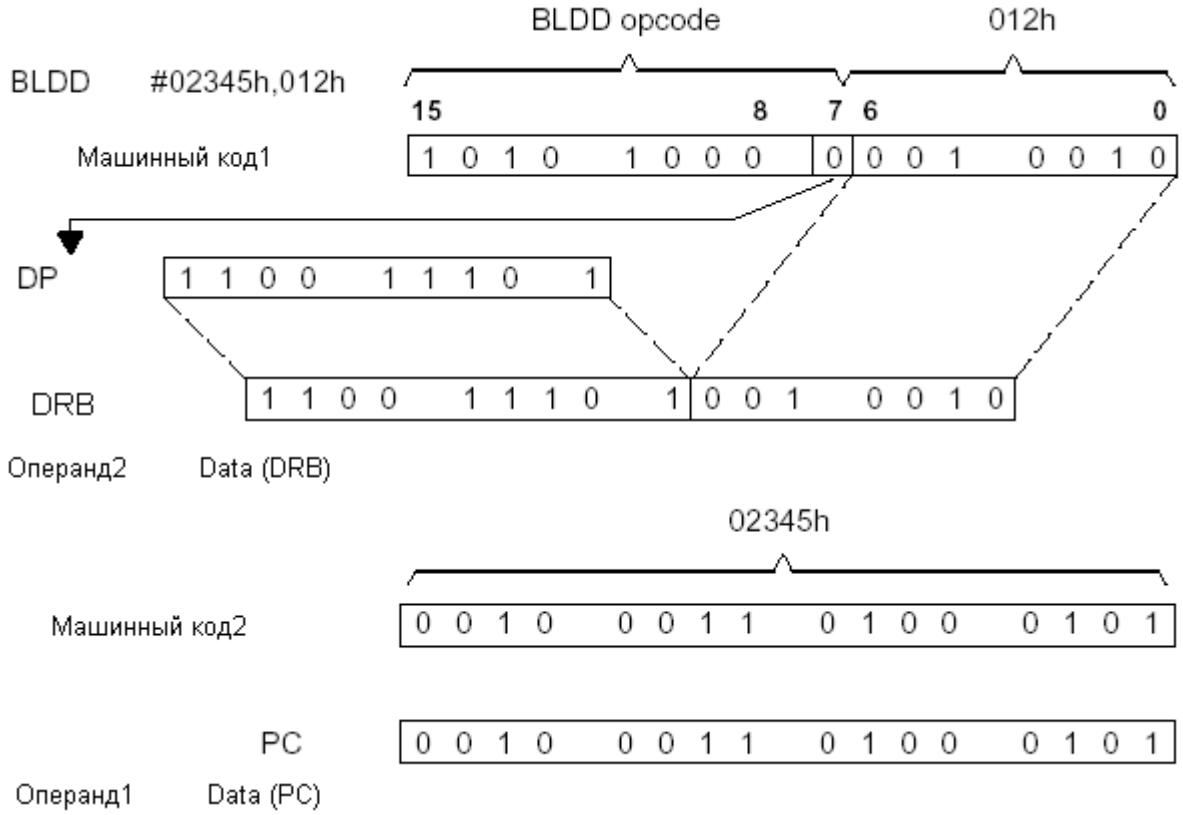


Рисунок 4.7 - Режим длинной непосредственной адресации с двумя операндами

Регистровый способ адресации работает подобно режиму длинной непосредственной адресации с тем исключением, что адрес содержится в одном из двух специализированных картированных в памяти регистров BMAR или DBMR. Преимущество этого способа заключается в том, что адрес блока памяти может быть изменен во время выполнения программы.

В инструкциях BLDD, BLDP и BLPD регистр BMAR используется в качестве указателя исходной или приемной области перемщения блока. Инструкции MADD и MADS тоже используют BMAR для адресации операнда, расположенного в программной памяти в операциях умножения с накоплением.

Рисунок 4.8 иллюстрирует использование BMAR в режиме регистровой адресации для однословной (в этом случае) разновидности инструкции BLDD. Также, как и в двухсловной BLDD (Рис. 4.7), адрес источника (OPERAND1) выбирается через PAB, а адрес места назначения (OPERAND2) генерируется в режиме прямой адресации.

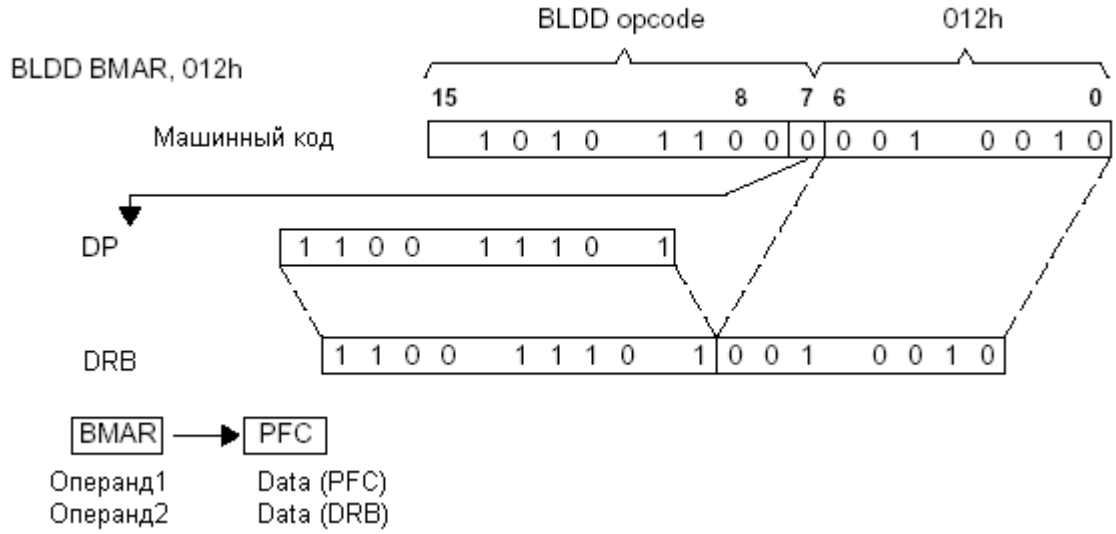


Рисунок 4.8 - Режим регистровой адресации с использованием BMAR

Аналогично, однословные (не использующие длинный непосредственный операнд) разновидности инструкций PLU: APL, CPL, OPL и XPL извлекают маску для выполнения логической операции из DBMR. Рисунок 4.9 иллюстрирует использование содержимого регистра DBMR в качестве маске AND для инструкции APL.

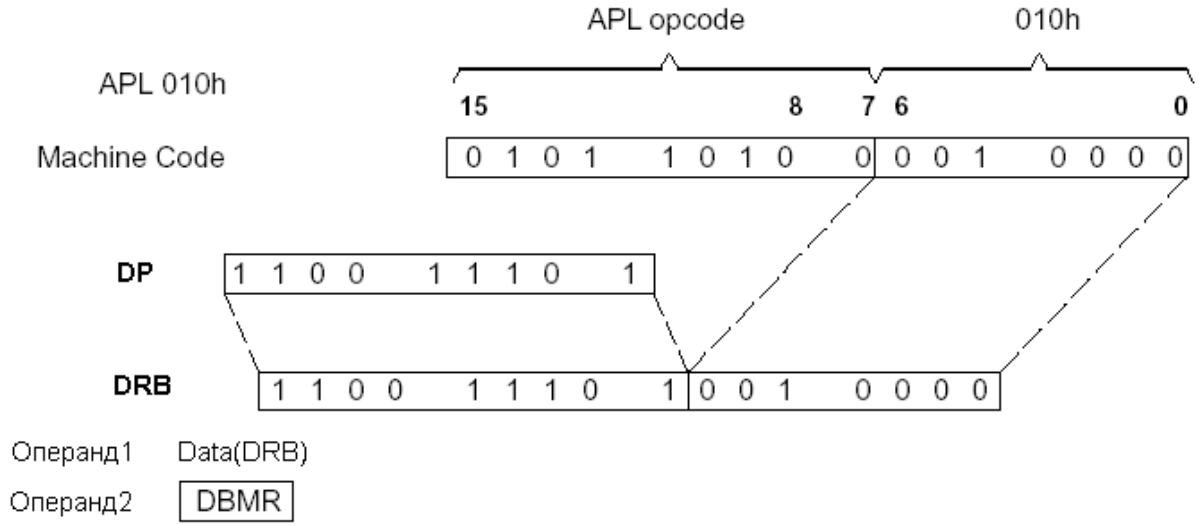


Рисунок 4.9 - Режим регистровой адресации с использованием DBMR

4.1.4.3 Вспомогательные регистры

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит восемь вспомогательных регистров (AR0 - AR7). Вспомогательные регистры могут использоваться для временного хранения информации, а сочетании с вычислительными устройствами процессора обеспечивают гибкую и мощную систему косвенной адресации памяти данных. При косвенной адресации любое из 64К слов пространства памяти данных может быть адресовано 16-ти

разрядным содержимым вспомогательного регистра. Чтобы выбрать один из восьми вспомогательных регистров, необходимо загрузить трехразрядный указатель вспомогательных регистров (ARP) значением от 0 до 7 в двоичном представлении, для ссылки на регистры от AR0 до AR7 соответственно (см. рисунок 4.10). Регистр, на который указывает ARP называется ***текущим*** и обозначается AR(ARP). Вспомогательные регистры могут быть загружены из памяти данных, из аккумулятора, из регистра результата произведения или непосредственным (коротким или длинным) операндом, содержащимся в коде соответствующей инструкции. Содержимое этих регистров можно сохранить в памяти данных или использовать в качестве операнда ALU. Вспомогательные регистры картированы в памяти по адресам, приведенным в таблице 4.2.

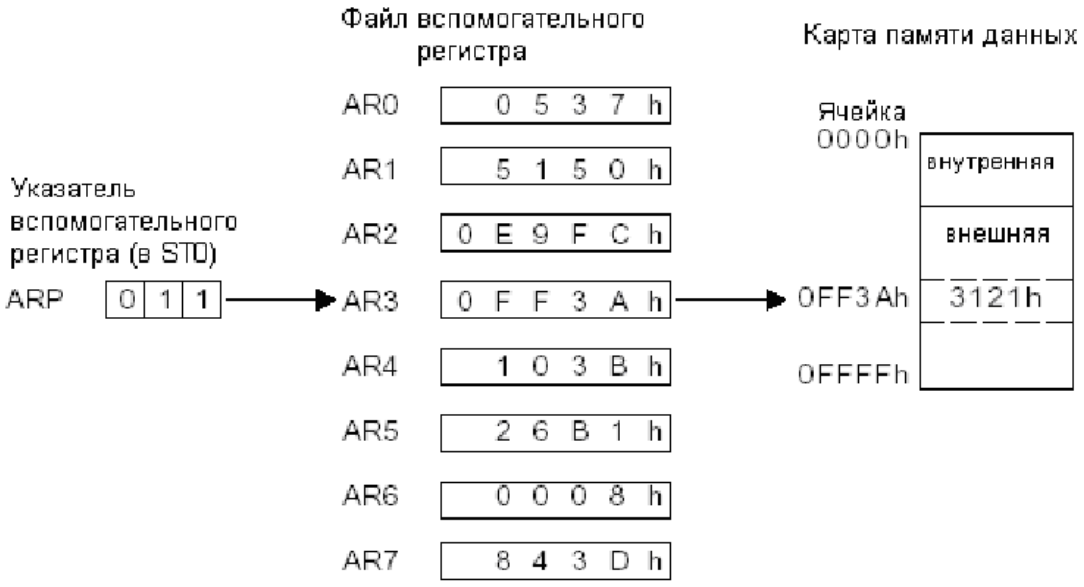


Рисунок 4.10 - Пример косвенной адресации с использованием вспомогательного регистра (AR3)

Файл вспомогательных регистров (AR0 - AR7) связан с арифметическим устройством вспомогательных регистров (ARAU), работающим с беззнаковой 16-ти разрядной двоичной арифметикой (см. Рисунок 4.11). Содержимое текущего вспомогательного регистра используется как адрес операнда в памяти данных После завершения доступа к памяти, содержимое текущего AR может быть модифицировано с помощью ARAU для генерации следующего адреса. Основные варианты модификации заключаются в прибавлении/вычитании из AR(ARP) единицы или содержимого регистра INDX (AR0 – в режиме совместимости). При этом наиболее распространенные варианты управления адресами не требуют центрального арифметико-логического устройства (CALU), освобождая его, таким образом, для других операций.

Если требуется более прогрессивное управление адресами, такое как адресация многомерных массивов, CALU может прямо читаться из / записываться в вспомогательные регистры. Однако, надо быть осторожным при записи из ALU во вспомогательный регистр, потому что перемещение ARAU из AR выполняется в течение фазы декодирования (второй цикл) конвейера, в то время как запись ALU происходит в течение фазы выполнения (четвертый цикл) конвейера. Следовательно, две команды, прямо следующие за записью ALU, не должны использовать вспомогательный регистр, записанный ALU.

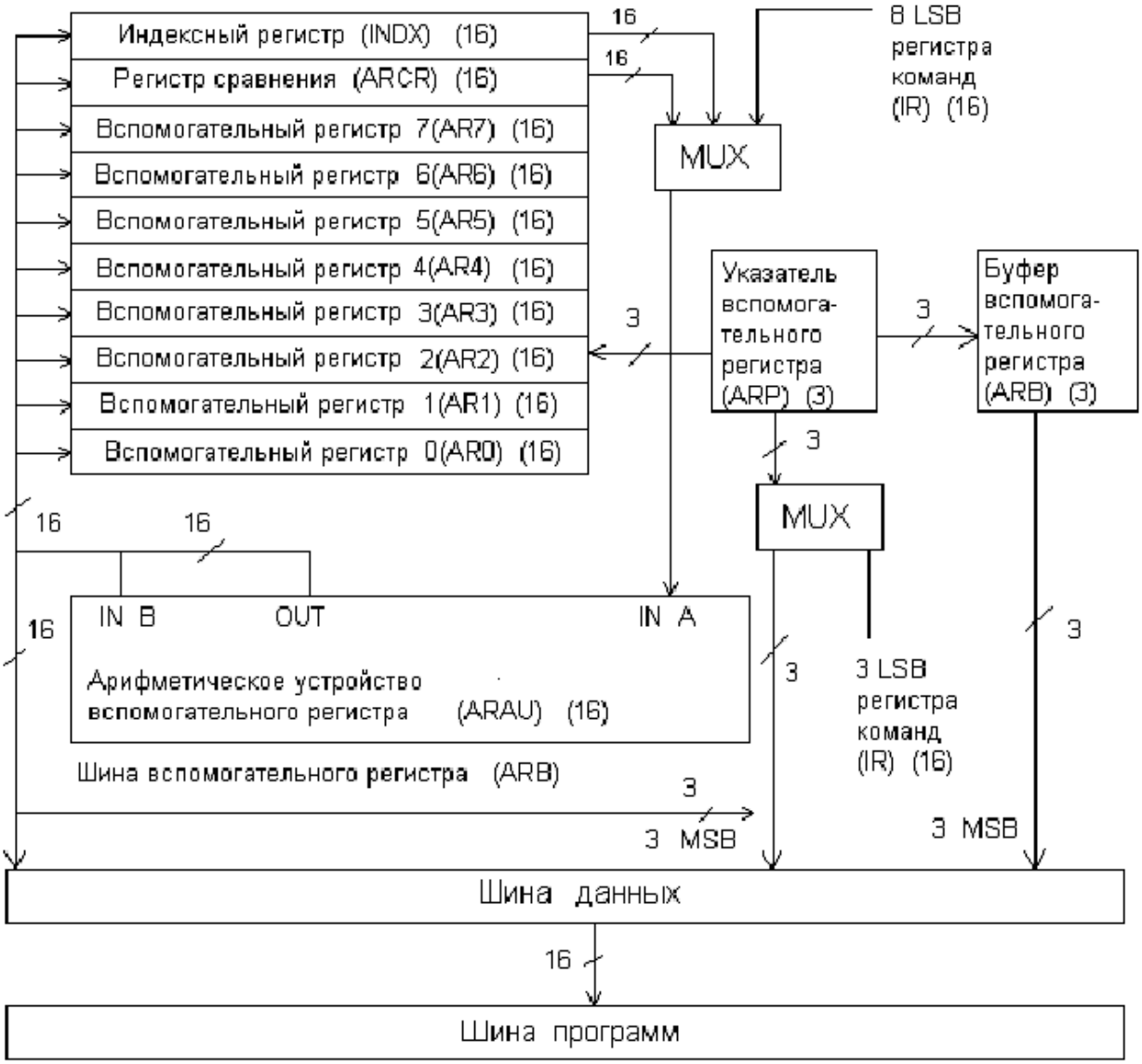


Рисунок 4.11 - Файл вспомогательных регистров

Как показано на Рисунке 4.11, один из входов ARAU может быть соединен либо с индексным регистром, либо с регистром сравнения, либо с младшими восемью разрядами

регистра инструкций. К другому вход подключен текущий AR (указываемый ARP). ARAU способно выполнять следующие операции над содержимым AR(ARP):

AR(ARP) + INDX → AR(ARP)	Индексация текущего AR путем прибавления к его содержимому беззнакового 16-разрядного числа, содержащегося в INDX
AR(ARP) - INDX → AR(ARP)	Индексация текущего AR путем вычитания из его содержимого беззнакового 16-разрядного числа, содержащегося в INDX
AR(ARP) + 1 → AR(ARP)	Увеличение текущего AR на единицу (инкремент)
AR(ARP) - 1 → AR(ARP)	Уменьшение текущего AR на единицу (декремент)
AR(ARP) → AR(ARP)	Текущий AR не модифицируется
AR(ARP) + IR(7-0) → AR(ARP)	Прибавление к содержимому текущего AR 8-разрядного непосредственного значения
AR(ARP) - IR(7-0) → AR(ARP)	Вычитание из содержимого текущего AR 8-разрядного непосредственного значения
AR(ARP) + rc(INDX) → AR(ARP)	Бит-реверсная индексация: сложение содержимого AR(ARP) и INDX с реверсивным распространением переноса (rc)
AR(ARP) - rc(INDX) → AR(ARP)	Бит-реверсная индексация: вычитание содержимого INDX из AR(ARP) с реверсивным распространением переноса (rc)
Если: (AR(ARP))=(ARCR), то TC=1 Если: (AR(ARP))<(ARCR), то TC=1 Если: (AR(ARP))>(ARCR), то TC=1 Если: (AR(ARP))≠(ARCR), то TC=1	Сравнение содержимого текущего AR и ARCR. В случае выполнения условия – установка бита TC в регистре состояния ST1 в единицу. Если условие не выполнено - обнуление бита TC.
Если (AR(ARP)) = (CBER), то AR(ARP) = CBSR	По дистижении конца циклического буфера, загрузка в AR(ARP) стартового адреса. Тестирование этого условия осуществляется перед выполнением модификации AR(ARP).

Индексный регистр (INDX) может быть прибавлен или вычтен к/из AR(ARP) в любом цикле изменения AR. Этот картированный в памяти 16-разрядный регистр используется для увеличения или уменьшения адреса при шаге больше 1 и применяется в операциях, таких как адресация вниз столбца матрицы. Регистр сравнения вспомогательных регистров (ARCR) используется как граница блоков данных. Инструкция CMRP поддерживает логическое сравнение AR(ARP) и ARCR. В процессорах 1867ВМ1 и 1867ВМ2 для функций индексации и сравнения используется регистр AR0.

Поскольку вспомогательные регистры являются картированными в памяти, косвенная адресация может быть реализована с помощью цетрального блока обработки данных (CALU). Например, умножитель может использоваться для вычисления адресов в трехмерных массивах. Нобходимо помнить, что из-за особенностей работы в конвейре, требуется задержка в два командных цикла перед тем, как вспомогательные регистры, модифицированные посредством CALU, можно будет использовать для адресации данных.

Регистры INDX и ARCR доступны для CALU, независимо от значения разряда NDX (т.е. инструкция SAMM ARCR всегда записывает только в ARCR).

Хотя основным назначением ARAU является генерация адреса данных, оно может также использоваться в качестве дополнительного арифметического устройства общего назначения, благодаря тому, что файл вспомогательных регистров непосредственно связан с памятью данных. ARAU выполняет 16-разрядную беззнаковую арифметику, тогда как CALU реализует 32-разрядную арифметику в коде с дополнением до двух. Команды BANZ и BANZD позволяют использовать вспомогательные регистры в качестве счетчиков цикла.

3-разрядный буфер указателя вспомогательного регистра (ARB), показанный на рисунке 4.11, обеспечивает сохранение содержимого ARP во время вызова подпрограмм.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) содержит два циклических буфера, управляемых посредством регистра CBCR. CBCR имеет следующий формат:

Разряд	Имя	Функция
0-2	CAR1	Указывает, какой вспомогательный регистр связан с циклическим буфером 1
3	CENB1	Включение (высоким уровнем) циклического буфера 1. При сбросе устанавливается в 0 (буфер выключен)
4-6	CAR2	Указывает, какой вспомогательный регистр связан с циклическим буфером 2
7	CENB2	Включение (высоким уровнем) циклического буфера 2. При сбросе устанавливается в 0 (буфер выключен)

Положительным фронтом сигнала сброса оба циклических буфера выключаются. Порядок активизации циклического буфера следующий: загрузка в CBSR 1/2 стартового, а в CBER 1/2 с конечного адреса буфера. Затем загрузка вспомогательного регистра, который будет использоваться с буфером, значением адреса между стартовым и конечным адресами. И, наконец, в CBCR загружается номер(а) соответствующего вспомогательного регистра(ов) и устанавливается разряд(ы) включения. По мере того, как адрес шаг за шагом проходит циклический буфер, значение вспомогательного регистра (перед очередной модификацией) сравнивается со значением, содержащимся в CBER. Если текущее значение вспомогательного регистра и CBER эквивалентны, значение, содержащееся в CBSR, автоматически загружается в AR. Если значения в CBER и вспомогательном регистре не эквивалентны, вспомогательный регистр модифицируется в соответствии с тем, как было предусмотрено в инструкции. Циклические буферы могут использоваться с такими модификациями как инкремент, или декремент. Если используется инкремент, то значение в CBER должно быть больше, чем значение в CBSR. Если используется декремент, значение в CBER должно быть меньше, чем значение в

CBSR. Могут использоваться и другие режимы косвенной адресации, однако ARAU тестирует только условие AR = CBER. ARAU не может тестировать момент перехода через границу циклического буфера при индексации вспомогательного регистра с шагом, отличным от единицы, когда предыдущее значение AR еще не достигло CBER, а следующее уже перешагнуло через величину, содержащуюся в CBER.

4.1.4.4 Блочные перемещения информации в памяти

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет команды для перемещения блоков данных и программ, позволяющие эффективно использовать пространство памяти.

Команда BLDD перемещает блоки в рамках памяти данных, команда BLPD перемещает блоки из памяти программ в память данных, а команда BLDP перемещает блоки из памяти данных в память программ. Очевидно, что эти инструкции должны содержать два адреса, определяющие источник и приемник данных. Один из адресов поступает из генератора адреса данных, в то время как другой берется или из длинной непосредственной константы, или из регистра BMAR. При использовании в режиме повтора (RPT и RPTZ), эти инструкции эффективно выполняют перемещения блоков во внутрикристальной или внешней памяти.

По аналогии с 1867ВМ2 в процессоре 1867ВЦ2Т (1867ВЦ2АТ) реализована возможность аппаратного перемещения данных во внутрикристальном ОЗУ по команде DMOV. DMOV копирует слово из адресованной в настоящий момент ячейки памяти данных во внутрикристальном ОЗУ в следующую ячейку (с адресом на 1 больше), в то время как данные из текущей ячейки используются в том же цикле (например, для CALU). Операция ARAU при использовании режима косвенной адресации также может выполнена в том же цикле. Команда DMOV может реализовывать алгоритмы, использующие операцию задержки z^{-1}, такие как свертка и цифровая фильтрация. Команда перемещения данных имеет максимальную эффективность при работе с внутрикристальной памятью двойного доступа. Операция перемещения данных в памяти однократного доступа, требует дополнительного цикла. Эта операция является непрерывной на границе блоков B0 и B1 памяти двойного доступа. Функция перемещения данных, реализованная в самостоятельной инструкции DMOV, является также составной частью более сложных (многофункциональных) инструкций: MACD (умножение с накоплением и перемещением данных; адрес коэффициента содержится в длинном непосредственном операнде), MADD (умножение с накоплением и перемещением данных; адрес коэффициента содержится в BMAR) и LTD (загрузка TREG0 с накоплением результатов и перемещением данных).

Команды TBLR/TBLW (чтение/запись таблиц) позволяют перемещать информацию между пространствами данных и программ. TBLR используется для чтения слов из памяти программ в память данных. TBLW используется для записи слов из памяти данных в память программ.

4.1.5 Центральное арифметико-логическое устройство (CALU)

Центральное арифметико-логическое устройство (CALU или блок обработки данных) процессора 1867ВЦ2Т (1867ВЦ2АТ) содержит 16-разрядный масштабирующий сдвиговый регистр, 16×16-разрядный параллельный умножитель, 32-разрядное арифметико-логическое устройство (ALU), 32-разрядный аккумулятор (ACC), 32-разрядный буфер аккумулятора (ACCB) и дополнительные сдвиговые регистры на выходах аккумулятора и умножителя. Этот подраздел описывает устройства вычислительной секции процессора и их функции. Рисунок 4.12 представляет собой блок-диаграмму, показывающую элементы CALU. Во время выполнения типичных команд CALU имеют место следующие действия:

1) данные выбираются из памяти по шине данных,

2) данные пересылаются через масштабирующий сдвиговый регистр в ALU, где выполняются арифметические операции,

3) результат перемещается в аккумулятор.

На один вход ALU данные всегда поступают с аккумулятора, а на другой могут подаваться из регистра результата (PREG) умножителя, буфера аккумулятора (ACCB) или из масштабирующего сдвигового регистра, загружаемого из памяти данных или аккумулятора (ACC).

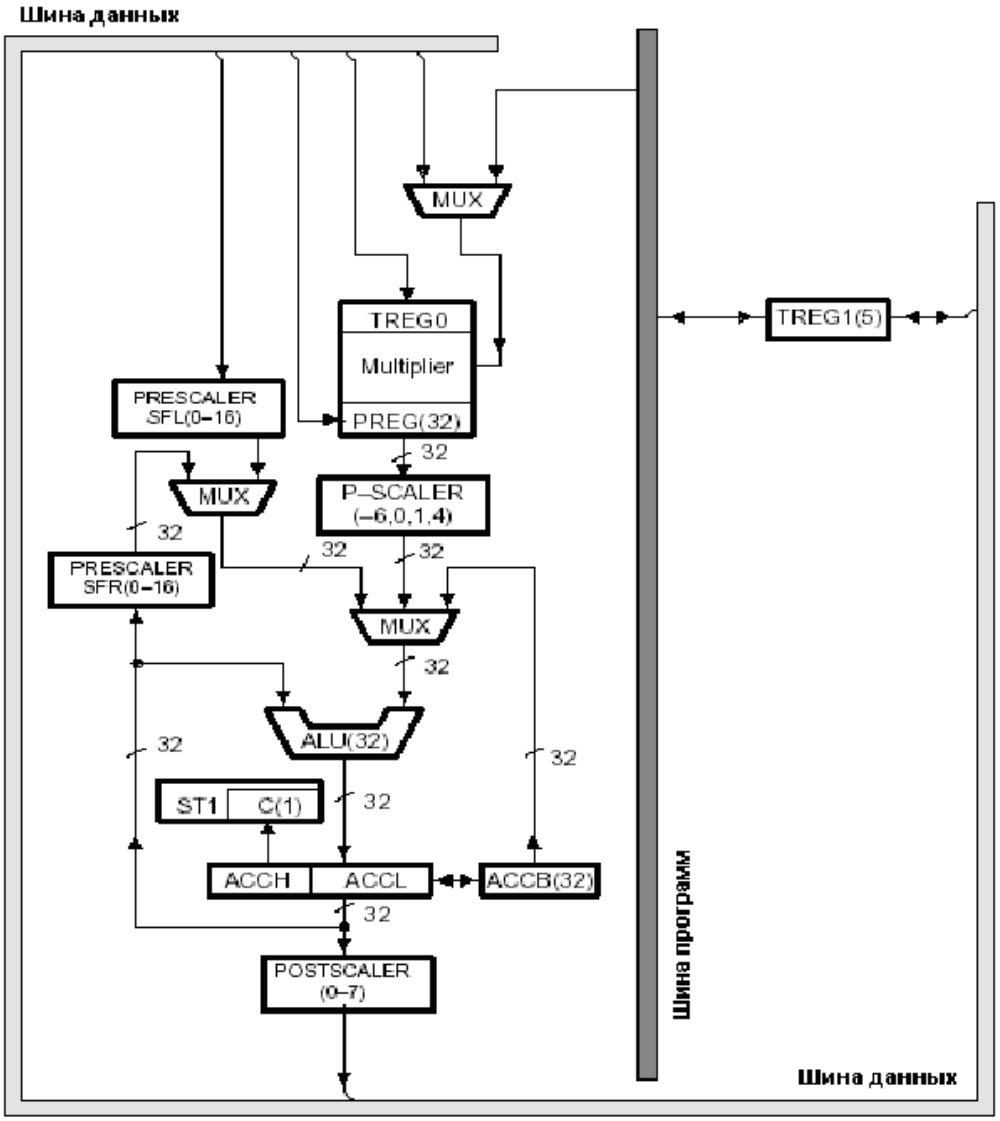


Рисунок 4.12 – Блок-диаграмма арифметико-логического устройства

4.1.5.1 Сдвиговый регистр предварительного масштабирования

В процессоре 1867ВЦ2Т (1867ВЦ2АТ) предусмотрен масштабирующий регистр сдвига, который имеет 16-разрядный вход, соединенный с шиной данных, и 32-разрядный выход, соединенный с ALU, как показано на рисунке 4.12. Масштабирующий сдвиговый регистр осуществляет сдвиг входных данных влево от 0 до 16 разрядов. Коэффициент сдвига определяется величиной, содержащейся в коде инструкции, или значением регистра TREG1. Младшие разряды выхода заполняются нулями; старшие разряды могут быть, в зависимости от состояния разряда SXM (режим расширения знака) регистра состояния ST1, либо заполнены нулями, либо значением знака (расширены на знак).

Процессор 1867ВЦ2Т (1867ВЦ2АТ) также содержит несколько других сдвиговых регистров, которые позволяют выполнять числовое масштабирование, выделение

разрядов, арифметику с повышенной точностью и предотвращение переполнения. Эти сдвиговые регистры соединены с выходом регистра результата и аккумулятора.

4.1.5.2 ALU и аккумулятор

32-разрядное ALU и аккумулятор процессора 1867ВЦ2Т (1867ВЦ2АТ) реализуют широкий спектр арифметических и логических функций, большинство из которых выполняется за один тактовый цикл. Вначале операция выполняется в ALU, затем результат пересылается в аккумулятор, где могут иметь место дополнительные операции, такие как сдвиг. Данные, входящие в ALU, могут масштабироваться сдвиговым регистром предварительного масштабирования.

ALU представляет собой арифметико-логическое устройство общего назначения, которое оперирует с 16-разрядными словами, считанными из памяти данных или с извлеченными из инструкций с непосредственными операндами. В дополнение к обычным арифметическим командам ALU может выполнять Булевы операции, обеспечивая возможность управления битами, требуемующуюся быстродействующему контроллеру.

На один вход ALU данные всегда поступают из аккумулятора, а на другой могут поступать из регистра результата (PREG) умножителя, буфера аккумулятора (ACCB) или с выхода масштабирующего сдвигового регистра (на вход которого, в свою очередь, поступают данные из памяти данных или из ACC).

После того, как ALU выполнило арифметическую или логическую операцию, результат сохраняется в аккумуляторе.

В следующем примере присвоим ACC = 0, PREG = 000222200h, PM = 00 и ACCB = 000333300h:

```
LACC #01111h,8    ;ACC = 00111100h. Загружает ACC из сдвигового регистра
                  ;предварительного масштабирования.
APAC              ;ACC = 00333300h. Прибавляет к ACC регистр результата.
ADDB              ;ACC = 00666600h. Прибавляет к ACC буфер аккумулятора.
```

32-разрядный аккумулятор (ACC) может быть разбит на два 16-разрядных сегмента (ACCH и ACCL) для сохранения в памяти данных, что представлено на рисунке 4.12. Сдвиговые регистры на выходе аккумулятора обеспечивают левый сдвиг от 0 до 7 бит. Этот сдвиг выполняется при выдаче информация на шину данных для сохранения в памяти. Содержимое аккумулятора при этом остается неизменным. Когда сдвиговый регистр последующего масштабирования используется со старшим словом аккумулятора (разряды 16 - 31), старшие разряды пропадают, а младшие разряды заполняются битами, сдвинутыми из младшего слова (разряды 0 - 15).

Когда сдвиговый регистр последующего масштабирования используется с младшим словом, младшие разряды заполняются нулями.

В следующем примере присвоим ACC = FF23 4567h:

```
SACL  TEMP1,7   ;TEMP1 = B380h.    ACC = FF23 4567h.
SACH  TEMP2,7   ;TEMP2 = 91A2h.    ACC = FF23 4567h.
```

Процессор 1867ВЦ2Т (1867ВЦ2АТ) поддерживает операции с плавающей запятой для применений, требующих большого динамического диапазона. Применение последовательных сдвигов влево в сочетании с инструкцией NORM (нормализация) позволяет нормализовать число с фиксированной запятой, содержащееся в аккумуляторе. Четыре разряда TREG1 определяют различные коэффициенты сдвига в регистре предварительного масштабирования для команд ADDT/LACT/SUBT (прибавить к/ загрузить в/ вычесть из аккумулятора со сдвигом, определенным TREG1). Эти команды могут быть использованы в арифметике с плавающей запятой, когда необходимо денормализовать число, т.е. преобразовать число в формате с плавающей запятой в представление с фиксированной запятой. Они также пригодны для автоматической регулировки усиления (AGC) на входе фильтра.

Одноцикловый 1 - 16-разрядный сдвиг аккумулятора вправо позволяет осуществлять эффективное выравнивание содержимого аккумулятора для арифметики с повышенной точностью. Этот сдвиг в сочетании с 32-разрядным буфером аккумулятора увеличивает эффективность ALU в операциях с арифметикой повышенной точности. Регистр буфера аккумулятора (ACCB) обеспечивает быстрое сохранение аккумулятора. ACCB может также использоваться в качестве входа ALU. Минимальное или максимальное значение в числовой последовательности может быть найдено путем сравнения содержимого ACCB и ACC. Такое сравнение выполняют инструкции CRLT (проверить, что ACC < ACCB) и CRGT (проверить, что ACC > ACCB). Минимальные и максимальные величины помещаются в эти регистры, и при выпонении условия разряд переноса (C) устанавливается в 1. Эти инструкции реализуют знаковые арифметические операции.

В следующем примере присвоим ACC = 012345678h и ACCB = 076543210h:

```
*  CRLT    ;ACC = ACCB = 1234 5678h. C = 1.
*  CRGT    ;ACC = ACCB = 7654 3210h. C = 0.
```

Режим обработки переполнения аккумулятора может быть включен/выключен путем установки/сброса разряда OVM регистра ST0. Когда режим обработки переполнения включен и переполнение произошло, выставляется флаг переполнения и происходит

насыщение аккумулятора, т.е. в него загружается (в зависимости от знака переполнения) максимально положительное или отрицательное значение. Значение аккумулятора при насыщении составляет 7FFF FFFFh (положительное) или 8000 0000h (отрицательное). Если разряд регистра состояния OVM (режим переполнения) очищен (=0) и произошло переполнение, результат операции загружается в аккумулятор без изменений (и, естественно, с потерей разрядов, вышедших за пределы длины аккумулятора). Режим обработки переполнения имеет смысл только для арифметических операций, т.к. логические операции (из-за отсутствия переноса) не могут вызвать переполнение результата.

В процессоре 1867ВЦ2Т (1867ВЦ2АТ) предусмотрены различные варианты инструкций ветвления (переходов), основанные на состоянии ALU и аккумулятора. Причем эти инструкции (например BCND) обеспечивают проверку множественных условий и ветвление происходит, когда все указанные условия выполнены. Инструкция BACC (переход по адресу, указанному в аккумуляторе) обеспечивает реализацию вычисляемых переходов (calculated GOTO). Инструкции тестирования разрядов (BIT и BITT) облегчают организацию условных переходов в зависимости от результата тестирования определенных разрядов (без их изменения) указанного слова в памяти данных.

Аккумулятор 1867ВЦ2Т (1867ВЦ2АТ) также имеет содержит разряд переноса, который устанавливается или сбрасывается в зависимости от результата выполнения различных операций в процессоре. Разряд переноса позволяет более эффективно выполнять вычисления с повышенной точностью. Это очень полезно при управлении переполнением. Разряд переноса зависит от большинства арифметических команд а так же, от команд одноразрядного циклического сдвига. Он не затрагивается при загрузке аккумулятора, выполнении логических операций или другими такими же неарифметическими командами или инструкциями управления. Примеры операций с разрядом переноса показаны на Рисунке 4.13.

Величина, прибавляемая к или вычитаемая из аккумулятора, может поступать с масштабирующего сдвигового регистра, с ACCB или с PREG. Разряд переноса устанавливается, если результат сложения или накопления вызывает перенос, или сбрасывается в нуль, если результат вычитания порождает отрицательный перенос (заем). В противном случае, он сбрасывается после сложения или устанавливается после вычитания.

В командах ADDC (прибавить к аккумулятору с переносом) и ADCB (прибавить ACCB к аккумулятору с переносом) используется предыдущее значение переноса. Инструкции SUBB (вычесть из аккумулятора с заемом) и SBBB (вычесть ACCB из

аккумулятора с заемом) используют логическую инверсию предыдущего значения переноса.

```
C  MSB           LSB                C  MSB           LSB
X   F F F F F F F F ACC              X   0 0 0 0 0 0 0 0 ACC
   +             1                    -             1
_____________________              _____________________
1   0 0 0 0 0 0 0 0                  0   F F F F F F F F

C  MSB           LSB                C  MSB           LSB
X   7 F F F F F F F ACC              X   8 0 0 0 0 0 0 1 ACC
   +             1 (OVM=0)            -             2 (OVM=0)
_____________________              _____________________
0   8 0 0 0 0 0 0 0                  1   7 F F F F F F F

C  MSB           LSB                C  MSB           LSB
1   0 0 0 0 0 0 0 0 ACC              X   F F F F F F F F ACC
   +             0 (ADDC)             -             1 (SUBB)
_____________________              _____________________
0   0 0 0 0 0 0 0 1                  1   F F F F F F F D
```

Рисунок 4.13 - Примеры операций разряда переноса

Одно исключение в работе разряда переноса, как показано на рисунке 4.13, заключается в использовании команды ADD с коэффициентом сдвига 16 (прибавить к старшему слову аккумулятора) и SUB со коэффициентом сдвига 16 (вычесть из старшего слова аккумулятора). Эти инструкции могут генерировать перенос или заем, но в отличие от стандартного варианта (см. выше) они не будут очищать перенос или заем в случае, если перенос или заем не генерируются. Это свойство облегчает реализацию арифметических вычислений с повышенной точностью.

Два условных операнда, C и NC, используются для реализации переходов, вызовов, возвратов и условных выполнений, на основе состояния разряда переноса. Для загрузки разряда переноса могут использоваться инструкции CLRC, LST #1 и SETC. Разряд переноса при аппаратном сбросе устанавливается в единицу.

Команды SFL и SFR (одноразрядный сдвиг влево/вправо) и команды ROL и ROR (циклический сдвиг влево/вправо) реализуют сдвиг или циклический сдвиг содержимого аккумулятора с прохождением данных через разряд переноса. SXM бит влияет на выполнение сдвига аккумулятора вправо инструкцией SFR. Когда SXM = 1, SFR выполняет арифметический сдвиг вправо, не меняя знак данных аккумулятора. Когда SXM = 0, SFR выполняет логический сдвиг, сдвигая с потерей младшие разряды и заполняя нулями старшие. Команда SFL (левый сдвиг аккумулятора) не зависит от состояния разряда SXM и ведет себя в обоих случаях одинаково, сдвигая с потерей старшие разряды и заполняя нулями младшие. Команды повтора (RPT и RPTZ) могут

использоваться с командами сдвига и циклического сдвига для реализации многоразрядных сдвигов.

65-разрядная комбинация аккумулятора, ACCB и разряда переноса может сдвигаться или вращаться, как описано выше, используя команды SFLR, SFRR, RORR и ROLR.

Аккумулятор может также быть сдвинут вправо на 0 - 31 разряд за два командных цикла или на 1 - 16 разрядов в одном цикле. Сдвинутые наружу младшие разряды (в количестве, соответствующем коэффициенту сдвига) теряются, а старшие разряды (в том же количестве) либо заполняются нулями, или копируют исходное (до сдвига) значение знакового разряда, в зависимости от состояния бита SXM. Коэффициент сдвига от 1 до 16 встраивается (при ассемблировании) в слово команды BSAR.

В следующем примере присвоим ACC = 1234 5678h:

```
BSAR   7      ;ACC = 02468ACEh.
```

Сдвиг вправо также может осуществляться через TREG1. Команда SATL сдвигает ACC на 0 – 15 разрядов, как определено 0 – 3 разрядами TREG1. Команда SATH сдвигает ACC на 16 разрядов вправо, если разряд 4 TREG1 равен 1. Приведенный ниже фрагмент последовательности кодов выполняет 0 – 31 - разрядный сдвиг ACC вправо, в зависимости от коэффициента сдвига, определенного как SHIFT.

В качестве примера присвоим SHIFT = 01Bh и ACC = 1234 5678h:

```
LMMR  TREG1, SHIFT    ;TREG1 = коэфф. сдвига 0 - 31. TREG1 = 1B.
SATH                  ;Если коэфф. сдвига > 15, то ACC >> 16
                      ;ACC = 00001234.
SATL                  ;ACC >> коэфф. сдвига. ACC = 0000 0002.
```

4.1.5.3 Умножитель, TREG0 и PREG

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет 16×16-разрядный аппаратный умножитель, который выдаёт вычисленный знаковый и беззнаковый 32-разрядный результат за один машинный цикл. Все команды умножения, за исключением команды MPYU (беззнаковое умножение), выполняют операцию умножения м учетом знака. Два умноженных числа обрабатываются как числа в двоичных кодах, а результатом является 32-разрядное число в двоичной системе исчисления.

Как показано на рисунке 4.12 следующие два регистра соединяются с умножителем:

− 16-разрядный временный регистр (TREG0), который содержит один операнд для умножителя,

− 32-разрядный регистр результата (PREG), который содержит результат.

Для PREG доступны четыре режима сдвига результата (PM). Эти режимы сдвига полезны при выполнении операций умножения/накопления, арифметики с дробями или выравнивания дробных результатов. Поле PM регистра состояния ST1 точно определяет режим сдвига PM, как показано в таблице 4.3.

Таблица 4.3 - Режимы сдвига результата

PM	Результирующий сдвиг
00	Нет сдвига
01	Левый сдвиг на 1 разряд
10	Левый сдвиг на 4 разряда
11	Правый сдвиг на 6 разрядов

Сдвиг на один разряд служит для компенсации добавочного знакового разряда, полученного при умножении двух 16-разрядных чисел в двоичных кодах (MPY). Четырехразрядный сдвиг используется в конъюнкции с командой MPY с коротким непосредственным значением (13 разрядов или меньше), чтобы исключить четыре добавочных знаковых разряда, полученных при умножении 16-разрядного числа на 13-разрядное число.

Вывод PREG может вместо того, чтобы сдвигаться вправо на 6 разрядов, включает выполнение возрастающих до 128 последовательных умножений/накоплений без переполнения.

Заметьте, что когда определяется правый сдвиг, результат всегда является знаково-расширенным, не считаясь со значением SXM.

Команда LT (загрузить TREG0) загружает TREG0, чтобы обеспечить один операнд (из шины данных), а команда MPY (умножение) обеспечивает второй операнд (также их шины данных). Умножение может также выполняться при помощи короткого или длинного непосредственного операнда, используя команду MPY с непосредственным операндом. Результат может получаться каждые два цикла, исключая случай, когда используется длинный непосредственный операнд. Четыре команды умножения/накопления (MAC, MACD, MADD и MADS) полностью используют вычислительную мощность умножителя, позволяя обоим операндам обрабатываться одновременно. Данные для этих операций могут передаваться в умножитель каждый цикл через шины программ и данных. Это обеспечивает одноцикловое умножение/накопление при использовании с командами повтора (RPT и RPTZ). В этих командах адреса коэффициентов генерируются посредством PC, в то время как адреса данных генерируются посредством ARAU. Это позволяет повторной команде последовательно

проходить значения из таблицы коэффициентов (PC+=) и переступать через данные в любых режимах косвенной адресации. Команда RPTZ также очищает аккумулятор и регистр результата для инициации операции умножения/накопления. Для примера рассмотрим умножение ряда одной матрицы на столбец второй матрицы. Для этого примера рассмотрим матрицы 10×10, MTRX1 указывает на начало первой матрицы, INDX = 10, а AR(ARP) указывает на начало второй матрицы:

```
RPTZ  #9          ;For i = 0,  i < 10,  i++
MAC   MTRX1, *0+ ;PREG = DATA (MTRX1 + i) x DATA [MTRX2 + (i x INDX)]
                  ;ACC += PREG.
APAC              ;ACC += PREG.
```

Команды MAC и MACD получают указатель коэффициента из длинного непосредственного адреса и являются, следовательно, командами, состоящими из двух слов. Команды MADS и MADD получают указатель коэффициента из BMAR и являются, следовательно, командами, состоящими из одного слова. Использование BMAR в качестве источника к таблице коэффициентов включает один блок кодов для поддержания множественных применений без необходимости модифицировать выполняемые коды, чтобы изменить длинный непосредственный адрес. Команды MACD и MADD также включают операцию перемещения данных (DMOV), которая в конъюнкции с выборкой сомножителя данных записывает значение данных в следующий более высокий адрес данных.

При повторе команды MACD и MADD поддерживают конструкции фильтров (взвешенные текущие средние), где по мере выполнения суммы результатов данные выборки сдвигаются в память, чтобы освободить пространство для следующей выборки и убрать старую выборку. Для примера ниже AR(ARP) указывает на самую старую выборку. BMAR указывает на таблицу коэффициентов. В дополнение к инициированию операции повтора команда RPTZ также очищает аккумулятор и регист результата. В этом примере PC хранится во временном регистре, пока выполняется команда повтора. Далее PC загружается со значением, хранящимся в BMAR. Шина программ используется для адресации коэффициентов и по мере повторного выполнения MADD. ARAU используется для генерации адреса данных выборки. Косвенная адресация с уменьшением используется здесь для шага данных выборки, начиная с самых старших адресов данных. По мере того, как данные выбираются, они также записываются в следующую по старшинству ячейку в памяти данных. Эта операция выравнивает данные для дальнейшего выполнения фильтра путем перемещения самой старшей выборки за прошлый конец массива выборки и создания пространства для следующей выборки в начале массива выборки. Предыдущий

результат (PREG) прибавляется к аккумулятору (ACC), пока умножаются два выбранных значения и в PREG загружается результат. Заметьте, что часть DMOV команд MACD и MADD не будет функционировать с внешними адресами памяти данных.

```
RPTZ   #9       ;ACC = PREG = 0. For I = 9 TO 0 Do
MADD   *-       ;SUM A_I x X_I. X_I+1 = X_I.
APAC            ;FINAL SUM.
```

Команда MPYU выполняет беззнаковое умножение, которое очень облегчает арифметические операции повышенной точности. Беззнаковое содержимое TREG0 умножается на беззнаковое содержимое адресной ячейки памяти данных, результат помещается в PREG. Это позволяет операндам больше 16 разрядов быть урезанными до 16-разрядных слов и обработанными раздельно, чтобы генерировать результаты больше 32 разрядов. Команды SQRA (возводить в квадрат/складывать) и SQRS (возводить в квадрат/вычитать) посылают то же значение в оба входа умножителя для возведения в квадрат значения памяти данных.

После умножения двух 16-разрядных чисел 32-разрядный результат загружается в 32-разрядный регистр результата (PREG). Результат из PREG может быть переслан в ALU или память данных через SPH (регистр результата хранения старшего слова) и SPL (регистр результата хранения младшего слова).

4.1.6 Блок программного управления

Блок программного управления содержит логическую схему декодирования и выполнения инструкций, управления конвейером ЦПУ, операций сохранения состояния ЦПУ и декодирования условных операций. Параллелизм архитектуры позволяет процессору 1867ВЦ2Т (1867ВЦ2АТ) выполнять три параллельных операции памяти в каком-либо машинном цикле: выборке инструкции, чтения операнда и записи операнда. Блок программного управления содержит следующие элементы:

- счётчик программ;
- регистры статуса и управления;
- аппаратный стек;
- устройство формирования адресов;
- регистр инструкций.

Следующие подразделы описывают функции каждого из этих компонентов.

4.1.6.1 Управление и генерация адреса программы

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет 16-разрядный счетчик программ и восьмиуровневый аппаратный стек для хранения PC. Счетчик программ адресует внешнюю и внутреннюю память программ в выбирающие команды. Стек используется в течение прерываний и подпрограмм.

Счетчик программ адресует память программ или внутрикристальную, или внешнекристальную через адресную шину программ (PAB). Через PAB команда адресуется в память программ и загружается в регистр команд (IR). Когда загружается IR, для старта цикла выборки следующей команды читается PC. PC может загружаться несколькими путями. Когда код выполняется последовательно, PC загружается при помощи PC + 1. Когда выполняется переход, PC загружается с длинным непосредственным значением, следующим прямо за командой перехода. В случае вызова подпрограммы PC помещается в стек, затем загружается с длинным непосредственным значением, следующим прямо за командой вызова. Команды возврата снимают со стека назад в PC для возврата к вызывающей или прерывающей последовательности кодов. В случае программного прерывания или прерывания PC загружается с адресом соответствующего вектора внутреннего прерывания. Содержимое аккумулятора может загружаться в PC для того, чтобы выполнять вычисленные операции GOTO. Это может выполняться, используя команды BACC (переход к адресу в аккумуляторе) или CALA (вызов подпрограммы в ячейке, определенной ACC).

Шина PAB может также использоваться для адресации данных, хранящихся в пространстве программ, или данных. Шина PAB используется таким образом, чтобы в повторенных командах второй операнд мог выбираться параллельно с шиной данных для двух-операндных операций. При повторе массив, адресованный посредством PAB, последовательно обращается через увеличение PC. Команды поблочной передачи (BLDD, BLDP и BLPD) используют обе шины таким образом, что при повторе конвейерная структура может считывать следующий операнд, в то время как записывается текущий. Команда BLPD загружает PC или с длинным непосредственным адресом, следующим за BLPD, или с содержимым адресного регистра перемещения блока (BMAR). Шина PAB затем используется для выборки исходных данных из пространства программ в этой операции перемещения блока. BLDP выполняет почти то же самое, исключая то, что шина PAB используется для операций адресата. Команда BLDD использует шину PAB для адресации пространства данных. Команды TBLR и TBLW действуют почти так же, как и команды BLDP и BLPD соответственно, исключая то, что PC загружается с младшими 16 разрядами аккумулятора вместо BMAR или длинного непосредственного адреса. Это

облегчает вычисленные операции табличного преобразования. Операции умножения/накопления (MAC, MACD, MADD и MADS) используют шину PAB для адресации таблицы их коэффициентов. Команды MAC и MACD загружают PC с длинным непосредственным адресом, следующим за командой. Команды MADD и MADS загружают PC с содержимым BMAR.

Для старта нового цикла выборки PC загружается или с PC + 1, или с адресом перехода (для таких команд как переходы, вызовы и прерывания). В случае условных переходов, когда операция перехода не берется, PC уменьшается еще раз выше ячейки непосредственного адреса перехода. В дополнение к условным переходам процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет широкое дополнение условных вызовов, выполнений и возвратов.

Эти команды выполняются, основываясь на следующих условиях:

Операнд	Условие	Описание
EQ	ACC = 0	Аккумулятор равен нулю
NEQ	ACC ≠ 0	Аккумулятор не равен нулю
LT	ACC < 0	Аккумулятор меньше нуля
LEQ	ACC ≤ 0	Аккумулятор меньше или равен нулю
GT	ACC > 0	Аккумулятор больше нуля
GEQ	ACC ≥ 0	Аккумулятор больше или равен нулю
C	C = 1	Перенос аккумулятора установлен в единицу
NC	C = 0	Перенос аккумулятора установлен на нуль
OV	OV = 1	Переполнение аккумулятора обнаружено
NOV	OV = 0	Переполнение аккумулятора не обнаружено
BIO	$\overline{BIO}$ низкий	Сигнал $\overline{BIO}$ находится на нижнем уровне
TC	TC = 1	Флаг тест/управление установлен на единицу
NTC	TC = 0	Флаг тест/управление установлен на нуль
UNC	нет	Безусловный переход

Повторные условия могут определяться в операндах условных команд. Если определяются повторные условия, должны встречаться все условия. Например,

```
BCND   BRANCH,LT,NOV      ;ACC < = 0 и не переполнен.
```

В этом примере должны встретиться оба условия (т.е. OV = 1 и ACC < 0) для взятого перехода. Условный переход BCND является командой, состоящей из двух слов. Условия для перехода не являются стабильными до четвертого цикла выполнения конвейера команд перехода, потому что предыдущая команда должна полностью выполнить разряды состояния аккумулятора. Следовательно, контроллер конвейера

останавливает декодировку команд, следующих за переходом до тех пор, пока допустимы условия. Если встречаются условия, определенные в операндах команды, то PC загружается со вторым словом, а ЦПУ с оперативной памятью начинает повторно заполнять конвейер командами по адресу перехода. Поскольку конвейер был заполнен, если принимается переход, команда перехода имеет эффективное время выполнения четырех циклов. Если, однако, какое-либо из условий не встречается, контроллер конвейера позволяет декодироваться следующей команде (уже выбранной). Это означает, если переход не принимается, эффективное время выполнения перехода составляет три цикла. Вызов подпрограммы может также выполняться условно. Команда CC действует подобно BCND, исключая то, что PC, указываювающий на следующую за CC команду, помещается в стек PC. Это вызывает возврат (посредством RET), чтобы снять со стека для возврата к вызывающей последовательности. Подпрограмма или функциональное устройство может иметь множественные пути доступа возврата, основанные на обработанных данных. Это устройство также поддерживает условные возвраты (RETC).

Например,

```
    CC OVER_FLOW,OV     ;Если переполнение, то выполнять
       .                ;программу операции над переполнением.
       .
       .
OVER_FLOW               ;Программа операции над переполнением.
       .
       .
       .
    RETC GEQ            ;Если ACC >= 0, то возврат.
       .
       .
       .
    RET                 ;Возврат.
```

В примере подпрограмма операции над переполнением вызывается, если главный алгоритм вызывает состояние переполнения. В течение подпрограммы проверяется ACC и, если возможно, подпрограмма возвращает к вызывающей последовательности. Если нет, перед возвратом необходима дополнительная обработка. Заметьте, что RETC, подобно RET, является командой, состоящей из единственного слова. Однако, из-за потенциальной прерывности PC, она еще действует с тем же эффективным временем выполнения как BCND и CC.

Чтобы избежать дополнительного цикла, 1867ВЦ2Т (1867ВЦ2АТ) имеет полный набор задержанных переходов, вызовов и возвратов. В задержанной операции переходов, вызовов или возвратов двукодовые слова, следующие за задержанной командой, выполняются до тех пор, пока выбирается команда или следующая за адресом перехода,

следовательно, не заполняя конвейер и давая эффективный двухцикловый переход. Если команда, следующая за задержанным переходом, является командой, состоящей из двух слов, то выполняться будет только она.

Например,

```
OPL       #030h,PMST
BCND      NEW_ADRS,EQ
```

или

```
BCNDD     NEW_ADRS,EQ
OPL       #030h,PMST.
```

Первый сегмент кода выполняется за шесть циклов (два для OPL и четыре для BCND). Второй сегмент кода занимает четыре цикла, потому что два потерянных цикла, следующих за BCNDD, заполняются с командой OPL. Заметьте, что условие, тестируемое на переходе, не затрагивается командой OPL, позволяя таким образом ему выполняться после перехода.

В случаях, где условный переход составляет одно или два слова кода, переход может быть заменен при помощи условной команды выполнения.

Например,

```
          BCND SUM, NC
          ADD  ONE
SUM       APAC
```

или

```
          XC   1,C
          ADD  ONE
          APAC
```

Первый сегмент кода занимает пять циклов. Второй сегмент кода занимает три цикла. Если условие встречается во втором сегменте кода, выполняется ADD. Если условие не встречается, то NOP вынужденно помещается в регистр команд над ADD. Условие должно быть стабильным в течение одного полного цикла перед выполнением команды XC. Это гарантирует выполнение условия перед командой, следующей за декодированием XC (модификации вспомогательного регистра происходят в течение фазы декодирования инструкции, таким образом команда должна останавливаться перед декодированием, чтобы обеспечить невыполнение этого). Следующие примеры показывают эту цикловую зависимость:

```
ZAC                       ;ACC = 0.
ADD   TEMP1               ;ACC = TEMP1.
XC    2,EQ                ;Если ACC = = 0,
SPL   #0EEEEh,TEMP2       ;то TEMP2 = EEEE.
```

или

```
ZAC                         ;ACC = 0.
ADD   #01234h               ;ACC = 00001234.
XC    2,EQ                  ;Если ACC = = 0,
SPLK  #0EEEEh,TEMP2         ;то TEMP2 = EEEE.
```

В первом сегменте кода TEMP2 = EEEE. Состояние NEQ, вызванное командой ADD, не устанавливается в то время, когда команда XC принимает решение. Следовательно, предыдущее условие EQ, вызванное командой ZAC, определяет условное выполнение. Поскольку это условие встречается, TEMP2 загружается командой SPLK. Во втором сегменте кода TEMP2 не устанавливается в EEEE. Состояние NEQ, вызванное командой ADD, устанавливается в течение одного полного цикла перед фазой выполнения XC из-за длинного непосредственного значения (#01234h), использованного в ADD, вызванного, чтобы составить двухцикловую команду. Поскольку условие не встречается, NOP вынужденно помещается выше обоих слов команды SPLK, состоящей из двух слов, и, следовательно, TEMP2 не затрагивается.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) также имеет возможность выполнения такта N + 1, где N - значение, загруженное в 16-разрядный регистр счетчика повтора (RPTC). Если используется средство повтора, команда выполняется, и RPTC уменьшается до тех пор, пока RPTC подходит к нулю. Это средство пригодно со многими командами, такими как NORM (нормализация содержимого аккумулятора), MACD (умножение и накопление с перемещением данных) и SUBC (условное вычитание). Когда освобождаются повторяющие команды, адрес программы и шины данных, параллельно с адресом данных и шинами данных выбирается второй операнд.

Это позволяет таким командам как MACD и BLKP при повторе эффективно выполняться в единственном цикле. Стек составляет 16 разрядов в ширину и восемь уровней в глубину. Стек PC доступен посредством использования команд PUSH и POP. Всякий раз когда содержимое PC помещается на верхушку стека, предыдущее содержимое каждого уровня спускается вниз, а нижняя (восьмая) ячейка стека пропадает. Следовательно, данные могут пропадать, если перед снятием со стека происходит больше восьми последовательных помещений в стек. Обратные перемещения происходят на операциях извлечения из стека. Любое извлечение из стека после семи последовательных извлечений выдает значение с нижнего уровня стека. Две дополнительные команды PSHD и POPD помещают значение памяти данных в стек или извлекают значение из стека в память данных. Эти команды позволяют стеку встраиваться в память данных для вложенности подпрограмм/прерываний сверх восьми уровней.

4.1.6.2 Работа конвейера

Конвейеризация команды состоит из последовательности шинных операций, которые встречаются во время выполнения инструкции. Конвейер команд: выбрать - декодировать - выбрать операнд - выполнить является прозрачным для пользователя, исключая некоторые случаи, когда конвейер может быть испорчен (такие как команды перехода). В работе конвейера операции выбора команды, декодирования, выбора операнда и выполнения являются независимыми, что допускает полное выполнение команды до перекрытия.

Таким образом, в течение любого взятого цикла могут активироваться от одной до четырех различных команд. Рисунок 4.14 показывает работу четырехуровнего конвейера для одноцикловых команд, состоящих из одного слова.

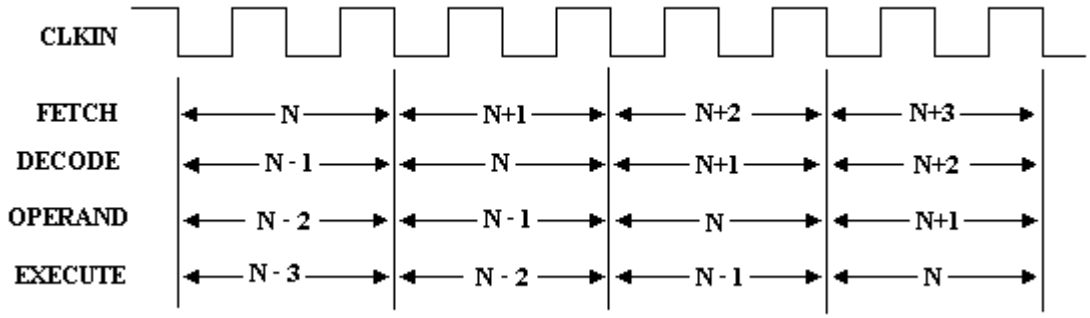


Рисунок 4.14 - Работа четырехуровневого конвейера

ARAU модифицирует вспомогательные регистры на выполнение в течение декодирования (второй фазы) конвейера. Это допускает генерацию адреса перед фазой выборки операнда. Однако, отображаемые в памяти доступы (т.е. SAMM, LMMR, SACL или SPLK) этих регистров происходят на фазе выполнения конвейера. Это значит, что следующие две команды после отображаемой в памяти загрузки вспомогательного регистра не должны использовать этот вспомогательный регистр. Следующие примеры кодов иллюстрируют действие отображаемой в памяти записи во вспомогательный регистр:

```
EXAM1  LAR    AR2,#67h    ;AR2 = 67h.
       LACC   #64h        ;ACC = 00000064.
       SAMM   AR2         ;Эта модификация отменяется посредством *-
                          ;модификаций на следующих двух командах.
       LACC   *-          ;AR2 = 66.
       ADD    *-          ;AR2 = 65.
```

или

```
EXAM2  LAR    AR2,#67h    ;AR2 = 67h.
```

```
        LACC    #64h        ;ACC = 00000064.
        SAMM    AR2         ;Модификация LACC *- происходит перед записью SAMM.
        LACC    *-          ;AR2 = 66.
        NOP                 ;AR2 = 64 (запись SAMM в AR2 происходит между
                            ;командами.
        ADD     *-          ;AR2 = 63.
```

или

```
EXAM3   LAR     AR2,#67h    ;AR2 = 67h.
        LACC    #64h        ;ACC = 00000064.
        SAMM    AR2         ;AR2 = 64.
        NOP                 ;Защита конвейера.
        NOP                 ;Защита конвейера.
        LACC    *-          ;AR2 = 63.
        ADD     *-          ;AR2 = 62.
```

В EXAM1 фаза декодирования команды ADD находится на таком цикле, как фаза выполнения (записи) инструкции SAMM. Обе эти команды пытаются загрузить AR2. Модификация ADD *- делает загрузку AR2, в то время как SAMM не выполняется. В EXAM2, чтобы избежать конфликта между модификацией ADD *- из AR2 и записью SAMM в AR2, NOP помещается стратегически. В этой последовательности кодов

```
AR2 = 67 -> 66 -> 64 -> 63
```

Заметьте, что адрес LACC базируется на значении в AR2 перед записью SAMM в AR2. В EXAM3 запись SAMM в AR2 заполняется перед тем, как либо LACC, либо ADD модифицирует AR2. В размещении двух инструкций NOP могут использоваться любые два слова команды, которые не модифицируют AR2. Это должны быть две команды, состоящие из одного слова или одна команда, состоящая из двух слов. Этот конвейерный эффект также применим к регистрам ARCR, BMAR, DMBR и INDX.

Команда NORM затрагивает AR(ARP) во время фазы выполнения из конвейера. В этом случае, как описывалось выше, работает управление каким-либо конвейером. Чтобы обнаружить модификацию и хранение (SAR) вспомогательного регистра непосредственно после команды NORM и автоматически разместить команду NOP для поддержания совместимости исходных программ с 1867ВМ2, был модифицирован ассемблер.

ЦПУ с оперативной памятью 1867ВЦ2Т (1867ВЦ2АТ) поддерживает реконфигурацию сегментов памяти как внутреннюю, так и внешнюю для устройства. Операции реконфигурации также происходят во время фазы выполнения конвейера. Следовательно, в списке должно быть два слова инструкции, следующие за командой, делающей реконфигурацию памяти, перед инструкцией, которая в настоящее время использует новую конфигурацию.

4.1.6.3 Регистры управления и состояния

Имеется четыре регистра состояния и управления для оперативной памяти процессора 1867ВЦ2Т (1867ВЦ2АТ). ST0 и ST1 содержат состояние различных условий и режимов процессора, совместимых с 1867ВМ2, в то время как PMST и CBCR содержат дополнительную информацию состояния и контроля для управления расширенными средствами оперативной памяти 1867ВЦ2Т (1867ВЦ2АТ). Эти регистры могут храниться в памяти данных и загружаться из памяти данных, таким образом предоставляя состоянию процессора сохраняться и восстанавливаться для подпрограмм. Каждый регистр ST0, ST1 и PMST имеет соединенный одноуровневый стек для автоматического контекстного сохранения при обработке прерывания. Стек автоматически выталкивается на возврате из прерывания.

Регистры PMST и CBCR располагаются в пространстве отображаемых в памяти регистров на нулевой странице пространства памяти данных. Следовательно, они могут активироваться непосредственно ALU и PLU. Они могут сохраняться тем же путем, что и любая другая ячейка памяти данных. Работа ALU и PLU меняет разряды этих регистров состояния во время фазы выполнения конвейера. Следующие два слова команды, идущие за модификацией этих регистров состояния, не могут затрагиваться реконфигурацией, вызванной изменением состояния.

ST0 и ST1 записываются, используя команду LST, и читаются, используя команду SST, (за исключением разряда INTM, который не затрагивается командой LST). В отличие от регистров PMST и CBCR, регистры ST0 и ST1 не располагаются в карте памяти и, следовательно, не могут управляться, используя команды PLU.

Индивидуальные разряды этих регистров могут быть установлены или очищены, используя инструкции SETC и CLRC. Например, знаково-расширенный режим устанавливается при помощи SETC SXM или очищается при помощи CLRL SXM.

Рисунок 4.15 показывает организацию четырех регистров состояния, указывая все разряды состояния, содержащиеся в каждом из них. Несколько разрядов в регистрах состояния резервируются и читаются как логические. Таблица 4.4 определяет все разряды состояния/управления.

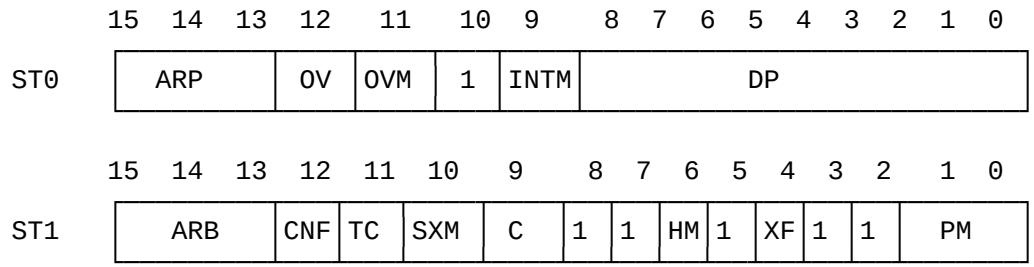


Рисунок 4.15 - Организация регистров состояния и управления

Таблица 4.4 - Определение поля регистра состояния

Поле	Функция
1	2
ARB	Буфер указателя вспомогательного регистра. Всякий раз как загружается ARP, старое значение ARP копируется в ARB, исключая во время инструкции LST. Когда ARB загружается через команду LST #1, такое же значение копируется также в ARP.
ARP	Указатель вспомогательного регистра. Это трехразрядное поле выбирает AR для использования при косвенной адресации. Когда загружается ARP, старое значение ARP копируется в ARB. ARP может модифицироваться инструкциями обращения к памяти при использовании косвенной адресации и командами LARP, MAR и LST. ARP также загружается при помощи такого же значения как и ARB, когда выполняется команда LST #1.
AVIS	Режим видимости адреса. Этот режим позволяет адресу внутренней программы появляться на контактах устройствах для разрешения трассировки адреса внутренней программы и разрешения декодирования вектора прерывания в конъюнкции с $\overline{\text{IACK}}$, где векторы прерывания располагаются во внутрикристальной памяти. Адрес внутренней программы выходит на контакты, где AVIS = 0. Линии адреса не меняются с внутренней программой, где AVIS = 1. Разряд AVIS при сбросе устанавливается на нуль.
BRAF	Активный флаг повтора блока. Этот разряд указывает активен ли в настоящий момент повтор блока. Запись нуля в этот разряд останавливает повтор блока. На сбросе BRAF устанавливается в нуль.
C	Разряд переноса. Этот разряд устанавливается в 1, если результат сложения генерирует перенос, или сбрасывается в нуль, если результат вычитания генерирует отрицательный перенос. В противном случае, он сбрасывается после сложения или устанавливается после вычитания, исключая, если инструкция - это ADD или SUB с 16-разрядным сдвигом. В этих случаях ADD может только устанавливать, а SUB только сбрасывать разряд переноса, но не может затрагивать его в другом случае. Команды одноразрядного сдвига и циклического сдвига также воздействуют на этот разряд, как и команды SETC, CLRC и LST #1. Команды перехода были предусмотрены для перехода на состояние C. На сбросе C устанавливается в 1.
CAR1	Вспомогательный регистр циклического буфера 1. Эти три разряда идентифицируют какой вспомогательный регистр присваивается циклическому буферу 1.

Продолжение таблицы 4.4

1	2
CAR2	Вспомогательный регистр циклического буфера 2. Эти три разряда идентифицируют какой вспомогательный регистр присваивается циклическому буферу 2.
CENB1	Включение циклического буфера 1. Когда этот разряд установлен в 1, включается циклический буфер 1. Когда установлен в 0, циклический буфер 1 отключается. CENB1 при сбросе устанавливается в нуль.
CENB2	Включение циклического буфера 2. Когда этот разряд установлен в 1, включается циклический буфер 2. Когда установлен в 0, циклический буфер 2 отключается. CENB2 при сбросе устанавливается в нуль.
CNF	Разряд управления конфигурацией внутрикристальным ОЗУ. Если установлен в 0, блоки ОЗУ двойного доступа реконфигурируемых данных отображаются в пространстве данных; в противном случае, они отображаются в пространстве программ. CNF может модифицироваться командами SETC CNF, CLRC CNF и LST #1. $\overline{RS}$ устанавливает CNF в 0.
DP	Указатель страниц памяти данных. 9-разрядный регистр DP конкатенируется с 7 младшими разрядами слова инструкции, чтобы образовать прямой адрес памяти из 16 разрядов. DP может быть модифицирован командами LST и LDP.
HM	Разряд режима HOLD. Когда HM = 1, процессор останавливает внутреннее выполнение при подтверждении активного HOLD. Когда HM = 0, процессор может продолжать выполнение из внутренней памяти программы, но помещает внешний интерфейс в третье состояние. При сбросе этот разряд устанавливается в 1.
INTM	Разряд режима прерывания. Когда установлен в 0, включаются все немаскированные прерывания. Когда установлен в 0, выключаются все маскируемые прерывания. INTM устанавливается и сбрасывается командами SETC INTM и CLRC INTM. $\overline{RS}$ и $\overline{IACK}$ также устанавливают INTM. INTM не воздействует на немаскируемые прерывания $\overline{RS}$ и $\overline{NMI}$. Заметьте, что INTM не затрагивается командой LST. Этот разряд при сбросе устанавливается в 1. Он также устанавливается в 1 при обработке маскируемых прерываний. Он сбрасывается в 0, когда выполняется RETE (возврат из прерывания с включением прерывания).
IPTR	Указатель вектора прерывания. Эти пять разрядов указывают на страницу 32 2К слов, где расположены векторы прерывания. Это позволяет пользователю преобразовать векторы прерывания в ОЗУ для операций загрузки. При сбросе все эти разряды устанавливаются в нуль. Следовательно, в пространстве памяти программы вектор сброса всегда располагается в нуле.
MP/$\overline{MC}$	Разряд микропроцессор/микрокомпьютер. Когда установлен в нуль, включается внутрикристальное ПЗУ. Когда установлен в единицу, внутрикристальное ПЗУ становится не адресуемым. Этот разряд при сбросе устанавливается в значение, соответствующее логическому уровню на контакте MP/$\overline{MC}$. Уровень на контакте MP/$\overline{MC}$ выбирается только при сбросе устройства и может не действовать до следующего сброса.

Продолжение таблицы 4.4

1	2
NDX	Дополнительный индексный регистр включения. Этот разряд распределяет индексную косвенную адресацию и вспомогательный адресный регистр для сравнения совместимости или с устройством TSM320C2x (NDX = 0), или расширенным режимом TSM320C5x (NDX = 1). Когда NDX = 0, команда LAR AR0 загружает регистры INDX и ARCR в дополнение к AR0. Это потому, что устройство TSM320C2x использует AR0 для индексации и операций сравнения AR. Когда NDX = 1, INDX и ARCR не затрагиваются инструкцией LAR. При сбросе NDX = 0.
OV	Разряд флага переполнения. Когда запирается сигнал переполнения, если переполнение происходит в ALU, OV устанавливается в 1. Как только происходит переполнение, OV остается установленным до тех пор, пока сброс BCND/D на OV/NOV или команда LST очистит OV.
OVLY	Разряд перекрытия ОЗУ. Этот разряд включает ячейки внутрикристального единичного доступа ОЗУ программы для картирования в пространство данных. Если установлен в единицу, блок памяти отображается в пространство данных. Если установлен в 0, блок памяти не является адресуемым в пространство данных. При сбросе этот разряд устанавливается в нуль.
OVM	Разряд режима переполнения. Когда установлен в 0, результаты переполнения переписываются обычно в аккумулятор. Когда установлен в 1, аккумулятор устанавливается или в наибольшее положительное, или наименьшее отрицательное значение. Инструкции SETC и CLRC устанавливают или сбрасывают этот бит соответственно. LST может также использоваться для модификаций OVM.
PM	Режим сдвига результата. Если эти два разряда составляют 00, 32-разрядный результат умножения не сдвигается при передаче в ALU. Если PM = 01, выход PREG сдвигается влево на одну позицию при передаче в ALU с заполнением LSB нулем. Если PM = 10, выход PREG сдвигается на четыре разряда при передаче в ALU с заполнением LSB нулями. PM = 11 обеспечивает сдвиг вправо на шесть разрядов со знаковым расширением. Обратите внимание, что содержимое PREG не изменяется. Сдвиг имеет место также при сохранении PREG в памяти данных. PM загружается командами SPM и LST #1. Разряды PM очищаются посредством $\overline{RS}$.
RAM	Включение ОЗУ программы. Этот разряд включает картирование блоков внутрикристального ОЗУ единичного доступа в пространство программ. Установленное в 1 ОЗУ отображает блок памяти в пространство программ. Установленное в 0 ОЗУ перемещает блок памяти из пространства программ. При сбросе этот разряд устанавливается в нуль.
SXM	Разряд режима знакового расширения. SXM производит расширение знака на данных, когда они пересылаются в аккумулятор через сдвигатель масштабирования. SXM = 0 запрещает знаковое расширение. SXM не затрагивает определений некоторых инструкций; например, команда ADDS запрещает расширение знака вне зависимости от SXM. Этот разряд устанавливается и сбрасывается командами SETC SXM и CLRC SXM, соответственно, а также может загружаться посредством LST #1. При сбросе этот разряд устанавливается в 1.

Продолжение таблицы 4.4

1	2
TC	Флаговый разряд тест/управление. Разряд TC затрагивается командами BIT, BITT, CMPR, LST #1, NORM, CPL, XPL, OPL и APL. Разряд TC устанавливается в 1, если разряд, тестируемый посредством BIT или BITT, равен 1, если состояние сравнения, тестируемое CMPR, существует между ARCR и другим AR, указанным ARP, если единственная функция OR двух старших разрядов аккумулятора верна при тестировании инструкцией NORM, если длинное непосредственное значение равно значению данных во время команды CPL, или если результат логической функции (XPL, OPL или APL) равен нулю. Команды условного перехода, вызова или возврата можно выполнять, базируясь на состоянии разряда TC.
TRM	Разряд управления Т-регистрами (TREG0, 1, 2). Этот разряд устанавливает 1867ВЦ2Т (1867ВЦ2АТ) в режим совместимости с 1867ВМ2 (TRM = 0) или расширенный режим для 1867ВЦ2Т (1867ВЦ2АТ) с использованием TREG0, 1, 2. Этот разряд затрагивает действие команд типа LT (LT, LTA, LTD, LTP и LTS). Процессор 1867ВМ2 использует Т-регистры как счетчик величины сдвига для сдвигателя с предварительным масштабированием и как адрес разряда в команде BITT. Когда TRM = 0, все команды типа LT записывают данные во все Т-регистры для поддержания совместимости исходных программ с устройством 1867ВЦ2Т (1867ВЦ2АТ). Когда TRM = 1, команды типа LT затрагивают только TREG0. При сбросе TRM = 0.
XF	Разряд состояния контакта XF. Этот разряд указывает состояние контакта XF, контакта вывода общего назначения. XF устанавливается и сбрасывается командами SETC XF и CLRC XF соответственно. При сбросе устанавливается в 1.

Таблица 4.5 - Управление конфигурацией внутрикристального ОЗУ 1867ВЦ2Т, 1867ВЦ2АТ

OVLY	RAM	Конфигурация
0	0	Внутрикристальное 9К ОЗУ отключено
0	1	Внутрикристальное 9К ОЗУ отображается в пространство программ
1	0	Внутрикристальное 9К ОЗУ отображается в пространство данных
1	1	Внутрикристальное 9К ОЗУ находится в пространствах программ и данных

4.1.6.4 Счетчик повтора

Счетчик повтора (RPTC) представляет собой 16-разрядный счетчик, который при загрузке числом N может повторить единичную инструкцию N + 1 раз. Регистр RPTC загружается командой RPT или RPTZ. RPTC очищается посредством сброса. Перед началом повтора следующей инструкции команда RPTZ очищает как ACC, так и PREG. Как только декодируется команда повтора (RPT или RPTZ), все прерывания (исключая

сброс) маскируются до окончания цикла повтора. Регистр RPTC располагается в пространстве отображамого в памяти регистра ЦПУ. Если в этот регистр пишется прикладная программа, то следующая инструкция, декодированная после того, как запишется регистр, будет повторяться значением в этом регистре. Необходимо избегать этой ситуации, поскольку последовательность кодов не защищена от прерываний.

Функция повтора должна использоваться с такими командами, как умножение/накопление (MAC и MACD), перемещение блока (BLKD и BLKP), переходы ввода/вывода (IN/OUT), чтение/запись таблицы (TBLR/TBLW). Эти инструкции, хотя обычно многоцикловые, при использовании средств повтора являются конвейерными и действительно становятся одноцикловыми командами. Например, команда чтения таблицы может иметь три или более циклов. Заметьте, что не все инструкции могут повторяться. Таблица 4.6 перечисляет все команды 1867ВЦ2Т, 1867ВЦ2АТ, разделенные по их повторяемости.

Таблица 4.6 - Повторяемость команд 1867ВЦ2Т, 1867ВЦ2АТ

Повторяемые команды	**Описание**
1	2
ADCB	Прибавить АССВ и бит переноса к АСС
ADD dma,shft	Прибавить к АСС, прямо адресованному со сдвигом
ADD *,shft	Прибавить к АСС, косвенно адресованному со сдвигом
ADDB	Прибавить АССВ к АСС
ADDC	Прибавить к АСС значение ячейки памяти и бита переноса с запрещением знакового расширения
ADDS	Прибавить к АСС значение ячейки памяти с запрещением знакового расширения
ADDT	Прибавить к АСС значение ячейки памяти со сдвигом, определенным TREG1
BSAR	Циклический сдвиг АСС вправо
NORM	Нормализовать АСС
ROL	Циклически сдвигать АСС влево на 1 бит
ROLB	Циклически сдвигать АСС и АССВ влево на 1 бит
ROR	Циклически сдвигать АСС вправо на 1 бит
RORB	Циклически сдвигать объединение АСС и АССВ вправо на 1 бит
SACH	Сохранить старшие разряды АСС со сдвигом
SACL	Сохранить младшие разряды АСС со сдвигом
SAMM	Сохранить младшие разряды АСС в нулевую страницу памяти данных
SATH	Сдвигать АСС вправо на 0-16 разрядов, как определено TREG1(4)

Продолжение таблицы 4.6

1	2
SATL	Сдвигать ACC вправо на 0-15 разрядов, как определено TREG1(0-3)
SBB	Вычесть ACCB из ACC
SBBB	Вычесть ACCB из ACC с отрицательным переносом
SFL	Сдвигать ACC влево на 1 бит
SFLB	Сдвигать объединенные ACC и ACCB влево на 1 бит
SFR	Сдвигать ACC вправо на 1 бит
SFRB	Сдвигать объединенные ACC и ACCB вправо на 1 бит
SUB dma,shft	Вычесть из ACC, прямо адресованного со сдвигом
SUB *,shft	Вычесть из ACC, косвенно адресованного со сдвигом
SUBB	Вычесть из ACC значение ячейки памяти данных и логическое инвертирование бита переноса с запрещением знакового расширения
SUBC	Условно вычесть из ACC
SUBS	Вычесть из ACC значение ячейки памяти данных с запрещением знака
SUBT	Вычесть из ACC значение ячейки памяти данных с левым сдвигом, определяемым TREG1
MAR	Модифицировать AR
SAR AR,*	Сохранить AR, косвенно адресованный
APL	AND DMBR с прямо/косвенно адресованным
OPL	OR DMBR с прямо/косвенно адресованным
XPL	XOR DMBR с ячейкой памяти данных и сохранение результата в памяти данных
APAC	Прибавить PREG к ACC
LTA	Загрузить TREG0 ячейкой памяти данных и прибавить PREG к ACC со сдвигом определяемым битами PM
LTD	Загрузить TREG0 ячейкой памяти данных и прибавить PREG к ACC со сдвигом определяемым битами PM и переместить данные
LTS	Загрузить TREG0 ячейкой памяти данных и вычесть PREG со сдвигом определяемым битами PM из ACC
MAC	Прибавить PREG со сдвигом определяемым битами PM к ACC; загрузить TREG0 ячейкой памяти данных и умножить значение ячейки памяти данных и памяти программ и сохранить результат в PREG
MACD	Прибавить PREG со сдвигом определяемым битами PM к ACC; загрузить TREG0 ячейкой памяти данных и умножить значение ячейки памяти данных и памяти программ и сохранить результат в PREG; переместить данные

Продолжение таблицы 4.6

1	2
MADD	Прибавить PREG со сдвигом определяемым битами PM к ACC; загрузить TREG0 ячейкой памяти данных и умножить значение ячейки памяти данных и значение определяемое BMAR и сохранить результат в PREG; переместить данные
MADS	Прибавить PREG со сдвигом определяемым битами PM к ACC; загрузить TREG0 ячейкой памяти данных и умножить значение ячейки памяти данных и значение определяемое BMAR и сохранить результат в PREG;
MPYA	Прибавить PREG со сдвигом определяемым битами PM к ACC и умножить значение ячейки памяти данных на TREG0 и сохранить результат в PREG
MPYS	Вычесть PREG со сдвигом определяемым битами PM из ACC и умножить значение ячейки памяти данных на TREG0 и сохранить результат в PREG
SPAC	Вычесть PREG сосдвигом определяемым битами PM из ACC
SPH	Сохранить старшие разряды PREG со сдвигом ределяемым битами PM в памяти данных
SPL	Сохранить младшие разряды PREG со сдвигом ределяемым битами PM в памяти данных
SQRA	Прибавить PREG со сдвигом определяемым битами PM к ACC; загрузить значение ячейки памяти данных в TREG0 и возвести в квадрат, сохранить результат в PREG
SQRS	Вычесть PREG со сдвигом определяемым битами PM из ACC; загрузить значение ячейки памяти данных в TREG0 и возвести в квадрат, сохранить результат в PREG
BLDD	Перемещение блока из памяти данных в память данных
BLDP	Перемещение блока из памяти данных в память программы
BLPD	Перемещение блока из программы данных в память данных
DMOV	Переместить в памяти прямо/косвенно адресованные данные на одну ячейку вверх
IN	Чтение из пространства ввода/вывода
OUT	Запись в пространство ввода/вывода
TBLR	Чтение из пространства программ в пространство данных
TBLW	Запись из пространства данных в пространство программ
NOP	Нет операции
POP	Поместить стек PC на младшие разряды ACC
POPD	Поместить стек PC до прямо/косвенно адресованного
PSHD	Вытолкнуть из стека прямо/косвенно адресованный в стек PC
PUSH	Вытолкнуть из стека младшие разряды ACC в стек PC
SST	Сохранит регистры состояния
Инструкции, не предназначенные для повторения	
ABS	Абсолютное значение ACC; бит переноса нулевой

Продолжение таблицы 4.6

1	2
AND	AND с младшими разрядами ACC; старшие разряды ACC нулевые
ANDB	AND ACCB с ACC
CMPL	Дополнение ACC
CRGT	Сохранить ACC в ACCB если ACC > ACCB
CRLT	Сохранить ACC в ACCB если ACC < ACCB
EXAR	Обменять ACC и ACCB содержимыми
LACB	Загрузить ACC в ACCB
LACC dma,shft	Загрузить ACC, прямо адресованный со сдвигом
LACC *,shft	Загрузить ACC, косвенно адресованный со сдвигом
LACL	Загрузить младшие разряды ACC значением ячейки памяти данных и нулевые старшие разряды ACC
LACT	Загрузить ACC значением ячейки памяти данных со сдвигом, определенным TREG1
LAMM	Загрузить младшие разряды ACC содержимым отображаемого в памяти регистра; старшие разряды ACC нулевые
NEG	Отрицательный ACC
OR	OR с младшими разрядами ACC значения ячейки памяти данных
ORB	OR ACCB с ACC
SACB	Сохранить ACC в ACCB
XOR	XOR с младшими разрядами ACC значения ячейки памяти данных
XORB	XOR ACCB с ACC
ZAP	Обнулить ACC и PREG
ZALR	Обнулить младшие разряды ACC, загрузить старшие разряды ACC с округлением
CMPR	Сравнить AR(ARP) с ARCR
LAR dma,AR	Загрузить AR, прямо адресованный
LAR *,AR	Загрузить AR, косвенно адресованный
LDP dma	ЗагрузитьDP, прямо адресованный
LDP *	Загрузить DP, косвенно адресованный
SAR AR,dma	Сохранить AR, прямо адресованный
CPL	Сравнить DBMR с прямо/косвенно адресованным
LPH	Загрузить старшие разряды PREG с прямо/косвенно адресованным
LT	Загрузить TREG0 с прямо/косвенно адресованным
LTP	Загрузить TREG0 прямо/косвенно и загрузить ACC из PREG
MPY	Умножить TREG0 прямо/косвенно
MPYU	Умножить TREG0 прямо/косвенно без знака
PAC	Загрузить ACC из PREG
SPM	Установить режим сдвига PREG
ZPR	Обнулить PREG
BIT	Тестировать разряд в слове данных
BITT	Тестировать разряд (определенный TREG2) в слове данных

Продолжение таблицы 4.6

1	2
CLRC	Очистить разряд состояния
LST	Загрузить регистры состояния
Неповторяемые инструкции	
ADD #(0h-0FFh)	Прибавить к ACC короткий непосредственный
ADD #(0100h-0FFFFh),shft	Прибавить к ACC длинный непосредственный со сдвигом
AND #(0h-0FFFFh),shft	AND с ACC длинный непосредственный со сдвигом
LACC #(0100h-0FFFFh),shft	Загрузить ACC длинный непосредственный
LACL #(0h-0FFh)	Загрузить ACC короткий непосредственный
OR #(0h-0FFFFh),shft	OR с ACC длинным непосредственным со сдвигом
SUB #(0h-0FFh)	Вычесть из ACC короткий непосредственный
SUB #(0100h-0FFFFh),shft	Вычесть из ACC длинный непосредственный со сдвигом
XOR #(0h-0FFFFh),shft	XOR с ACC длинным непосредственным со сдвигом
ZAC	Обнулить ACC
ADRK	Прибавить к ACC короткое непосредственное
LAR #(0h-0FFFFh),AR	Загрузить AR с длинным непосредственным
LDP #(0h-01FFh)	Загрузить DP короткий непосредственный
SBRK	Вычесть из AR короткий непосредственный
APL #(0h-0FFFFh)	AND длинный непосредственный с прямо/косвенно адресованным
CPL #(0h-0FFFFh)	Сравнить длинный непосредственный с прямо/косвенно адресованным
OPL #(0h-0FFFFh)	OR длинный непосредственный с прямо/косвенно адресованным
SPLK #(0h-0FFFFh)	Сохранить длинный непосредственный прямо/косвенно адресованному
XPL #(0h-0FFFFh)	XOR длинный непосредственный прямо/косвенно адресованным
B[D]	Безусловный [задержанный] переход
BACC[D]	Переход [задержанный] к адресу, определенному в младших разрядах ACC
BANZ[D]	Переход [задержанный] на AR(ARP) не нуль
BCND[D]	Условный [задержанный] переход
CALA[D]	Вызов [задержанный] к адресу, определенному в младших разрядах ACC
CALL[D]	Вызов [задержанный] подпрограммы
CC[D]	Условный вызов [задержанный] подпрограммы
RET	Возврат из подпрограммы
RCND[D]	Возврат [задержанный] из подпрограммы условно
RETE	Возврат из сервисной программы прерывания с общим автоматическим включением
RETI	Возврат из сервисной программы прерывания
TRAP	Прерывание программного обеспечения
XC	Выполнять следующую инструкцию условно
LMMR	Загрузить отображаемый в памяти регистр
SMMR	Сохранить отображаемый в памяти регистр
IDLE	Бездействующее ЦПУ
RPT	Повторять следующую инструкцию N + 1 тактов
RPTB	Повторять блок

Продолжение таблицы 4.6

1	2
RPTZ	Обнулить ACC и PREG и повторять следующую инструкцию N + 1 тактов

4.1.6.5 Функция повторения блока

Средство повторения блока обеспечивает выполнение заполненного нулями цикла для реализации циклов FOR и DO. Эта функция управляется тремя регистрами (PASR, PAER и BRCR) и разрядом BRAF в регистре PMST. Регистр счетчика повторения блока (BRCR) загружается со счетом цикла от 0 до 65535. Затем, выполняется инструкция RPTB (блок повторения), загружая таким образом стартовый регистр адреса программы (PASR) с адресом инструкции, следующей за командой RPTB, и загружая конечный регистр адреса программы (PAER) при помощи длинного непосредственного операнда. Длинный непосредственный операнд является адресом инструкции, следующей за последней командой в цикле минус один. Разряд BRAF автоматически устанавливается активным посредством выполнения инструкции RPTB, таким образом, стартует цикл PAER, сравнивается с PC при каждом изменении PC. Если они равны, BRCR уменьшается. Если BRCR больше или равен нулю, PASR загружается в PC, таким образом, начиная цикл снова. Если нет, разряд BRAF устанавливается нулевым, а процессор завершает выполнение оставшегося цикла кодов.

Например,

```
    SPLK   #0Fh,BRCR    ;Установить счет цикла до 16.
    RPTB   END_LOOP-1   ;Для I = BRCR; I >= 0,I--.
*
    ZAP                 ;ACC = PREG = 0.
    SQRA   *,AR2        ;PREG = X².
    SPL    SQRX         ;Сохранить X².
    MPY    *            ;PREG = b x X.
    LTA    SQRX         ;ACC = bX.  TREG = X².
    MPY    *            ;PREG = aX².
    APAC                ;ACC = aX² + bX.
    ADD    *,0,AR3      ;ACC = aX² + bX + c = Y.
    SACL   *,0,AR1      ;Сохранить Y.
    CRGT                ;Сохранить MAX.
END_LOOP
```

Пример реализует 16 выполнений уравнения $Y = aX^2 + bX + c$ и сохраняет максимальное значение в ACCB. PAER загружается с адресом последнего слова в кодовом сегменте. Метка END_LOOP размещается после последней инструкции, и инструкция RPTB длинная непосредственная определяется как END_LOOP-1 в случае последнего слова в цикле в инструкции, состоящей из двух слов.

Имеется только один набор регистров повторения блока, таким образом, множественные повторения блока не могут быть встроены без сохранения контекста внешнего блока или используя BANZD. Самый простой способ выполнения встроенных циклов заключается в использовании RPTB только для глубокого цикла и, используя BANZD, для всех внешних циклов.

Это, однако, значительные операции сохранения цикла, поскольку глубокий цикл повторяется значительно больше времени, чем внешние циклы.

Повторения блоков могут встраиваться посредством сохранения контекста внешнего цикла перед инициацией глубокого цикла, затем восстанавливая контекст внешнего цикла после завершения глубокого цикла. Сохранение и восстановление контекста показано в следующем примере:

```
   SMMR BRCR,TEMP1      ;Сохранить счетчик повторения блока.
   SMMR PASR,TEMP2      ;Сохранить начальный адрес блока.
   SMMR PAER,TEMP3      ;Сохранить конечный адрес блока.
   SPLK #NUM_LOOP,BRCR ;Установить счет глубокого цикла.
   RPTB END_INNER       ;Для I = 0; I <= BRCR; I++.
     .
     .
     .

END_INNER
   LMMR BRCR,TEMP1      ;Восстановить счетчик повторения блока.
   OPL  #1,PMST         ;Установить BRAF для продолжения внешнего цикла.
   LMMR PASR,TEMP2      ;Восстановить начальный адрес блока.
   LMMR PAER,TEMP3      ;Восстановить конечный адрес блока.
```

В этом примере операции сохранения и восстановления контекста занимают 14 циклов. Обратите внимание, что BANZ/BANZD и единственный повторяемый циклы также могут находиться внутри повторения блока. Повторяемые коды могут также быть включены в вызовы подпрограмм. При возврате повторение блока заканчивается. Повторяемые блоки прерываемы. Когда в рамках повторяемого блока кодов случается разрешенное прерывание, ALU внутренне прерывает прерывание, а при возврате ISR заканчивается повторение блока.

Пока блок повторяется, активный флаг повторения блока (BRAF) регистра PMST устанавливается в единицу. Этот флаг устанавливается посредством выполнения инструкции RPTB и сбрасывается, когда PC = PAER и BRCR = 0. Этот флаг может быть очищен и/или сброшен через регистр PMST. Цикл WHILE может быть реализован, используя инструкцию RPTB и условно сбрасывая разряд BRAF. Следующий пример кодов очищает BRAF таким образом, что процессор будет бросать цикл кодов и, если произойдет переполнение, продолжать последовательно обращаться к последним в конце цикла инструкциям.

```
XC    2,OV          ;Если переполнение,
APL   #0FFFFh,PMST  ;то выключить повторение блока.
```

Аналогично цикл WHILE может быть реализован посредством установки разряда BRAF в нуль, если встречается условие выхода. Если это имеет место, программа завершает текущую передачу через цикл, но не возвращается к началу. Разряд должен быть установлен у наименьших трех инструкций перед концом цикла для выхода из цикла. Циклы повторения блока могут выходить и возвращаться без остановки и перезапуска цикла.

Переходы, вызовы и прерывания не обязательно воздействуют на цикл. Когда управление программой возвращается к циклу, работа цикла заканчивается. Следующий пример иллюстрирует повторение блока с небольшим циклом кодов, который выполняет последовательность задач. Задачи хранятся в таблице, адресованной посредством TEMPOF. Число задач для выполнения определяется в NUM_TASKS.

```
   BLPD   NUM_TASKS,BRCR    ;Установить счет цикла.
   SPLK   #(TASKS-1),TEMPOF ;TEMPOF указывает на список задач.
   RPTB   ENDCALL-1         ;Для I = 0, I <= NUM_TASKS, ++.

TASK_HANDLER
   LACC   TEMPOF            ;ACC указывает на таблицу задач.
   ADD    #1                ;Указатель шага к следующей задаче.
   SACL   TEMPOF            ;Сохранить для следующей передачи цикла.
   TBLR   TEMPOE            ;Прочитать адрес задачи.
   LACC   TEMPOE            ;ACC = адресу задачи.
   CALA                     ;Задача вызова.
ENDCALL
```

Установленный для примера счетчик повторения блока (BRCR) загружается с числом задач для выполнения минус 1. Далее, адрес таблицы задачи загружается во временный регистр. Повторение блока начинается с выполнения инструкции RPTB. Регистр PASR загружается с адресом инструкции LACC TEMPOF. Регистр PAER загружается с адресом последнего слова таблицы. Обратите внимание, что метка, отмечающая конец цикла, размещается за последней инструкцией, затем PAER загружается с этой меткой минус 1. Метка должна располагаться перед инструкцией CALA, затем загрузить PAER с адресом метки, поскольку она является командой, состоящей из одного слова. Однако, если последняя инструкция в этом цикле состояла из двух слов, второе слово инструкции читаться не будет и длинный непосредственный операнд должен заместиться на первую инструкцию в цикле.

Внутри цикла указатель к таблице задачи увеличивается и сохраняется. Затем, адрес задачи читается из таблицы и загружается в аккумулятор. Далее, задача вызывается инструкцией CALA. Обратите внимание, что когда задача возвращается к указателю

задачи, она возвращается к началу цикла. Это происходит потому, что PC уже был загружен с PASR перед тем, как CALA прерывно выполняет PC. Следовательно, когда выполняется CALA, адрес начала цикла помещается в стек PC.

Циклы могут быть построены посредством сохранения содержимого BRCR, PAER и PASR до начала выполнения внутреннего цикла и восстанавливая его на окончании внутреннего цикла и располагая разряд BRAF, т.к. он занимает общих 14 циклов для сохранения (6 циклов) и восстановления (8 циклов) регистров повторения блока, наименьшие внешние циклы могут быть реализованы при помощи метода зациклившегося BANZD, который берет два дополнительных цикла на цикл. Одноцикловые инструкции могут в рамках повторения блока повторяться, используя инструкции RPT и RPTZ.

4.1.6.6 Режим пониженной мощности

Работая в режиме пониженной мощности, оперативная память 1867ВЦ2Т, 1867ВЦ2АТ входит в состояние бездействия и потребляет значительно меньше мощности по сравнению с нормально работающим устройством. Режим пониженной мощности вызывается или посредством выполнения инструкций или устанавливая ввод HOLD в низкий уровень при помощи установки в единицу разряда состояния HM.

Инструкция IDLE2 используется для полного закрытия системы ЦПУ с оперативной памятью так же, как и всех внутрикристальных периферийных устройств. Поскольку внутрикристальные периферийные устройства останавливаются при режиме пониженной мощности, они не могут использоваться в генерации прерывания для запуска устройства, как описано выше в режиме IDLE. Однако, мощность значительно понижается потому, что устройство полностью останавливается. Режим пониженной мощности завершается активацией любого из выводов внешнего прерывания ($\overline{RS}$, $\overline{INT1}$, $\overline{INT2}$, $\overline{INT3}$, $\overline{INT4}$).

Когда режим пониженной мощности инициируется через сигнал $\overline{HOLD}$ (HM = 1), ЦПУ останавливает адрес, и линии управления преключаются в третье состояние для дальнейшего понижения мощности. Когда режим пониженной мощности инициируется через сигнал $\overline{HOLD}$, то он заканчивается, когда ввод $\overline{HOLD}$ становится неактивным. Если HM = 0, имеется еще некоторое понижение мощности системы путем выключения сигнальных драйверов управления адресом и памятью. Если в настоящий момент не требуется внешних доступов в систему, это также может использоваться. Использование $\overline{HOLD}$ не останавливает работу внутрикристальных периферийных устройств (т.е.

внутрикристальные таймеры и последовательные порты продолжают работать, несмотря на уровень $\overline{\text{HOLD}}$ или состояние разряда HM).

4.1.7 Параллельное логическое устройство (PLU)

Параллельное логическое устройство (PLU) предназначено для прямой установки, очистки, тестирования или переключения различных разрядов в регистре управления/состояния или любой ячейке памяти данных. PLU (блок-диаграмма показана на рисунке 4.16) обеспечивает передачу логических операций к значениям памяти данных, не воздействуя на содержимое аккумулятора или регистра результата. Оно может использоваться для установки или очистки разрядов в регистре управления или тестирования разрядов во флаговом регистре.

PLU выполняет операцию чтения-модификации-записи данных, сохраненных в пространстве данных. Работа PLU начинается с выбора одного операнда из пространства памяти данных и выбора второго из/или длинного непосредственного на шине программ, или регистра управления динамическим разрядом (DBMR). Затем PLU выполняет логическую операцию, определенную инструкцией, на двух операндах. Результат записывается в какую-либо ячейку памяти данных, из которой был выбран первый операнд.

PLU допускает прямую манипуляцию разрядами в любой ячейке в пространстве памяти данных. Эта прямая манипуляция битами осуществляется посредством инструкций AND, OR, XOR или загружая 16-разрядное длинное непосредственное значение в ячейку данных.

Например, чтобы инициализировать регистр управления циклическим буфером (CBCR), используя AR1 для циклического буфера 1 и AR2 для циклического буфера 2, но не включая циклического буфера, требуется выполнить:

```
SPLK #021h,CBCR  ;Сохранить периферийный длинный непосредственный. (DP = 0).
 Для дальнейшего разрешения циклического буфера 1 или 2, выполнить:
OPL #088h,CBCR   ;Установить разряд 7 и разряд 3 в CBCR.
```

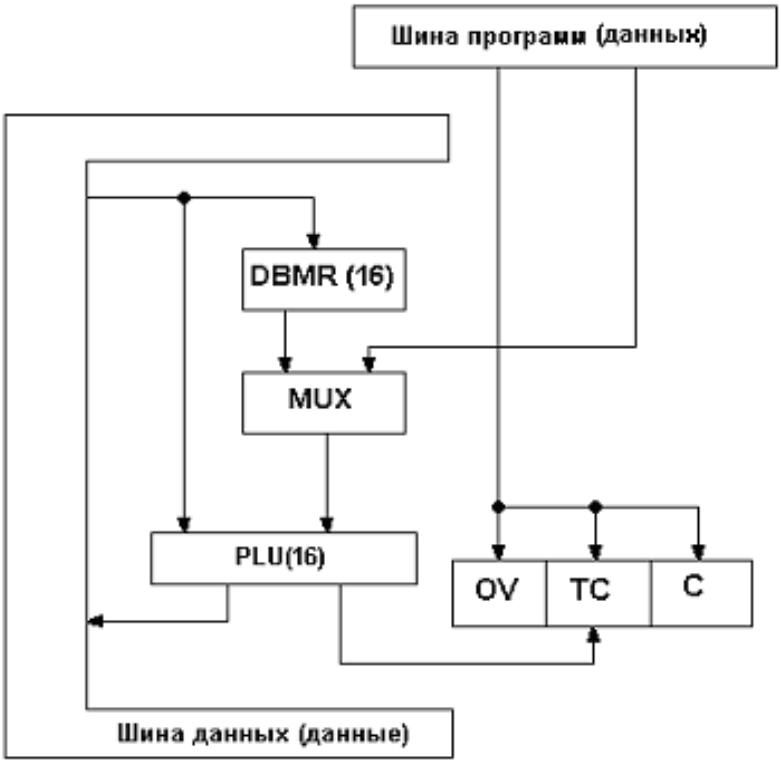


Рисунок 4.16 - Блок-диаграмма параллельного логического устройства

Тестирование для отдельного разряда в определенном регистре или слове данных еще осуществляется через инструкцию BIT; однако, может быть тестирован против образца при помощи инструкции CPL (сравнение параллельного длинного непосредственного). Если значение данных эквивалентно длинному непосредственному значению, то разряд TC устанавливается в 1. Разряд TC устанавливается, если результатом любой инструкции PLU является нуль.

Функции установки, очистки и переключения бита могут также выполняться при помощи 16-разрядного значения динамического регистра вместо длинного непосредственного значения. Это осуществляется при помощи следующих трех инструкций: AND регистр DBMR и данные (APL), OR регистр DBMR и данные (OPL) и XOR регистр DBMR и данные (XPL).

Разряд TC в ST1 также устанавливается инструкциями APL, OPL и XPL, если результат работы PLU (значение, записанное назад в память данных) равняется нулю. Это позволяет разрядам тестироваться и очищаться одновременно.

Например

```
APL     #0FF00h,TEMP          ;Очистить младший байт и проверить для установки
                              ;разрядов в старшем байте.
BCND    HIGH_BITS_SET,NTC     ;Если разряды активны в старшем байте, то переход.
```

Или

```
XPL     #1,TEMP          ;Переключить разряд 0.
BCND    BIT_SET,TC       ;Если разряд был установлен, переход.
                         ;Если нет, разряд тотчас установить.
```

В первом примере младший байт слова флага очищается, в то время как старший байт заменяется на любые активные флаги (разряды = 1). Если ни один из флагов в старшем байте не устанавливается, то окончательная операция APL выдает нуль TEMP и устанавливает разряд TC в 1. Если какой-либо из флагов устанавливается в старший байт, то окончательная операция APL выдает ненулевое значение TEMP и устанавливает разряд TC в 0. Следовательно, условный переход, следующий за инструкцией APL, переходит, если какой-либо из разрядов в старшем байте не нуль. Второй пример тестирует флаг. Если нулевой, он устанавливается в единицу; если единица, он очищается и берется переход. Инструкции PLU могут работать в каком-либо месте пространства адреса данных, таким образом они могут использоваться для работы и с флагами, сохраненными в ячейках ОЗУ, и регистрами управления как для внутри-, так и внекристальных периферийных устройств.

4.1.8 Прерывания

Оперативная память процессора 1867ВЦ2Т (1867ВЦ2АТ) имеет шестнадцать маскируемых пользователем прерываний ($\overline{\text{INT16}} - \overline{\text{INT1}}$). ЦПУ с оперативной памятью проектируется для поддержания до 16 маскируемых пользователем прерываний. 1867ВЦ2Т (1867ВЦ2АТ) использует только девять из этих прерываний (другие внутри устройства подключены к высокому уровню). Прерывания могут вызываться последовательными портами (RINT и XINT), таймером (TINT) и командой программного прерывания (TRAP). Прерывания приоритезируются при помощи сброса (RS), имеющий наибольший приоритет, и INT15, имеющий наименьший приоритет.

4.1.8.1 Сброс

Сброс ($\overline{\text{RS}}$) является немаскируемым внешним прерыванием, которое может быть использовано в любое время, чтобы привести 1867ВЦ2Т (1867ВЦ2АТ) в известное состояние. Сброс обычно применяется после подачи питания, когда процессор находится в неизвестном состоянии. Появление низкого уровня сигнала $\overline{\text{RS}}$ заставляет процессор 1867ВЦ2Т (1867ВЦ2АТ) прекратить выполнение и сбросить счетчик программы до нуля.

$\overline{RS}$ воздействует на различные регистры и разряды состояния. При подаче питания состояние процессора неопределённое. Для правильной работы системы после подачи питания сигнал сброса должен быть установлен низким для минимум шести тактовых циклов. Устройство будет закрыто для импульса сброса и будет достаточно долго генерировать внутренний импульс сброса для гарантии сброса устройства. Работа процессора начинается с ячейки 0, которая обычно содержит команду перехода к программе инициализации системы.

При получении сигнала $\overline{RS}$ происходит следующее:

1) Логический 0 загружается в разряд CNF (управление конфигурацией) в регистре состояния ST1, отображающемся в блоке 0 ОЗУ двойного доступа в пространстве адреса данных.

2) Счетчик программы (PC) устанавливается в 0. Адрес шины (линии A15-A0) неизвестен, пока $\overline{RS}$ является низким, если ввод $\overline{HOLD}$ устройства не низкий. В этом случае адресные линии помещаются в третье состояние до тех пор, пока $\overline{HOLD}$ не примет обратно высокое состояние.

3) Все прерывания отменяются установлением разряда INTM (режим прерывания) в 1; заметьте, что $\overline{RS}$ является немаскируемым. Флаговый регистр прерывания (IFR) очищается.

4) Разряды состояния:

```
0 -> OV, 1 -> XF, 1 -> SXM, 0 -> PM, 1 -> HM, 0 -> BRAF,
0 -> TRM, 0 -> NDX, 0 -> CENB1, 0 -> CENB2, 0 -> IPTR,
0 -> OVLY, 0 -> AVIS, 0 -> RAM, 0 -> BIG, 0 -> CNF,
```

```
1 -> INTM, MP/MC (Контакт) -> PMST (MP/MC) и 1 -> C.
```

Обратите внимание, что оставшиеся разряды состояния остаются не определенными и, соответственно, должны быть инициализированы.

5) Регистр распределения глобальной памяти (GREG) очищается, чтобы сделать всю память локальной.

6) Очищается счетчик повтора (RPTC).

7) Сигнал $\overline{IACK}$ (подтверждение приема прерывания) генерируется каким-либо способом как маскируемое прерывание.

8) Сигнал синхронизированного сброса ($\overline{SRESET}$) подстраивается к периферийным цепям для их инициализации. Работа начинается из ячейки 0 памяти программы, когда сигнал $\overline{RS}$ становится высоким. Обратите внимание, если $\overline{HOLD}$

устанавливается в то время, когда устанавливается и $\overline{RS}$, нормальная работа сброса происходит внутренне; но все шины и линии управления остаются в третьем состоянии. Когда освобождается $\overline{\text{HOLD}}$ и $\overline{\text{RS}}$, работа начинается с нулевой ячейки.

4.1.8.2 Работа прерываний

Этот подраздел подробно описывает организацию и управление прерываниями. Условные векторные ячейки и приоритеты для всех внутренних и внешних прерываний показаны в таблице 4.7.

Таблица 4.7 - Приоритеты и ячейки прерываний

Имя	Ячейка		Приоритет	Функция
	Dec	Hex		
$\overline{\text{RS}}$	0	0	1(наибольший)	Сигнал сброса
$\overline{\text{INT1}}$	2	2	3	Пользовательское прерывание #1
$\overline{\text{INT2}}$	4	4	4	Пользовательское прерывание #2
$\overline{\text{INT3}}$	6	6	5	Пользовательское прерывание #3
$\overline{\text{INT4}}$	8	8	6	Пользовательское прерывание #4
$\overline{\text{INT5}}$	10	A	7	Пользовательское прерывание #5
$\overline{\text{INT6}}$	12	C	8	Пользовательское прерывание #6
$\overline{\text{INT7}}$	14	E	9	Пользовательское прерывание #7
$\overline{\text{INT8}}$	16	10	10	Пользовательское прерывание #8
$\overline{\text{INT9}}$	18	12	11	Пользовательское прерывание #9
$\overline{\text{INT10}}$	20	14	12	Пользовательское прерывание #10
$\overline{\text{INT11}}$	22	16	13	Пользовательское прерывание #11
$\overline{\text{INT12}}$	24	18	14	Пользовательское прерывание #12
$\overline{\text{INT13}}$	26	1A	15	Пользовательское прерывание #13
$\overline{\text{INT14}}$	28	1C	16	Пользовательское прерывание #14
$\overline{\text{INT15}}$	30	1E	17	Пользовательское прерывание #15
$\overline{\text{INT16}}$	32	20	18	Пользовательское прерывание #16
TRAP	34	22	N/A	Вектор инструкции TRAP
$\overline{\text{NMI}}$	36	24	2	Немаскируемое прерывание

Инструкция TRAP (программное прерывание) не приоритезирована, но включена сюда, т.к. она имеет свою собственную векторную ячейку. Каждый адрес прерывания был расположен отдельно в двух ячейках так, что инструкции перехода могут размещаться в этих ячейках. Чтобы сделать вектор, сохраненный в ПЗУ, перепрограммируемым, могут использоваться следующие коды:

```
LAMM TEMP0     ;ACC = адресу ISR.
BACC           ;Переход к ISR.
```

TEMP0 располагается в B2 и задерживает адрес сервисной программы прерывания (ISR). Обратите внимание, что адреса ISR должны загружаться в B2 перед разрешением прерываний.

Векторы прерывания могут переотображаться к началу любой страницы 2К-слова в памяти программ. Адрес вектора прерывания генерируется конкатенацией разряда IPTR из PMST с номером вектора прерывания (1 - 16), сдвинутым на один, как показано на рисунке 4.17.

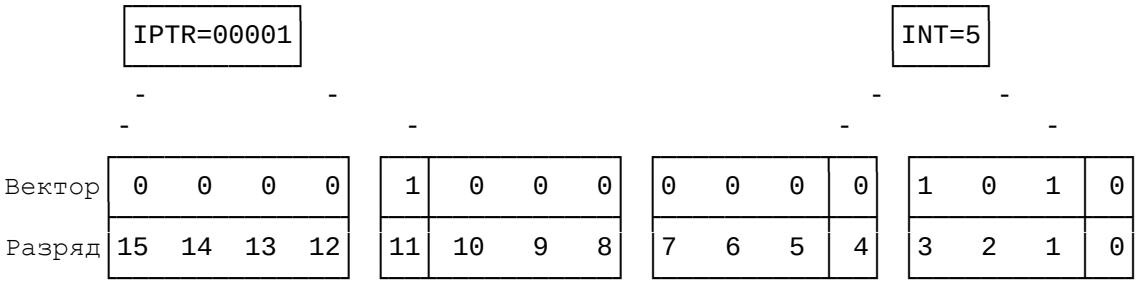


Рисунок 4.17 - Генерация адреса вектора прерывания

При сбросе все разряды IPTR устанавливаются в нуль, таким образом отображая векторы к нулевой странице в пространстве памяти программ. Это значит, что вектор сброса всегда располагается у нуля. Векторы прерывания могут перемещаться к другой ячейке посредством загрузки ненулевого значения в разряды IPTR. Например, векторы прерывания для запуска могут быть перемещены к ячейке 0800h, загружая IPTR единицей.

Если происходит прерывание, в 16-разрядном флаговом регистре прерывания (IFR) активируется флаг. Каждое прерывание сохраняется в IFR до тех пор, пока оно узнается ЦПУ. Любое из следующих четырех событий будет очищать флаг прерывания:

1) сброс устройства ($\overline{RS}$ является активным низким), или
2) программа принимает внутреннее прерывание, или
3) программа записывает единицу в соответствующий разряд в IFR, или
4) выполнение инструкции INTR с соответствующим числом прерываний.

IFR располагается по адресу 06h в пространстве памяти данных и может быть считан, чтобы идентифицировать активные прерывания, и записан, чтобы очистить прерывания. Регистр IFR располагается следующим образом:

15-12	11	10 9	8	7	6	5	4	3	2	1	0
Reserved	HINT	Reserved	$\overline{\text{INT4}}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{\text{INT3}}$	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$

Логическая единица в положении разрядов IFR указывает на ожидание прерывания. Например, если IFR читается в 0005h, то активны $\overline{\text{INT3}}$ и $\overline{\text{INT1}}$. Единица может записываться в особый разряд, чтобы очистить соответствующее прерывание. В примере, если единица записывается в разряд 0 (0001h для IFR), то прерывание $\overline{\text{INT1}}$ должно быть очищено. Все ожидающиеся прерывания должны очищаться путем записи текущего содержимого IFR обратно в IFR. В примере выше значение 0005h должно быть записано назад в IFR, чтобы очистить оба ожидающихся прерывания. Следующий пример очищает эти два вектора, не затрагивая любые другие флаги, которые могли быть установлены:

```
SPLK #5, IFR      ;Очистить флаги для INT1 и INT3.
```

Когда берется соответствующее внутреннее прерывание, флаг прерывания автоматически очищается. Когда ЦПУ принимает прерывание, оно заполняет шину инструкций командой INTR. Эта команда вталкивает PC в соответствующий адрес и выбирает программно-управляемый вектор. Пока выбирается первое слово программно-управляемого вектора, он генерирует сигнал подтверждения прерывания $\overline{\text{IACK}}$, который очищает соответствующий флаговый разряд прерывания. Номер определенного прерывания, будучи взятым, указывается адресными разрядами A1 - A4 на заднем фронте сигнала $\overline{\text{IACK}}$. Обратите внимание, что устройство должно работать в режиме адресной видимости (AVIS = 0), если векторы прерывания для декодирования располагаются во внутрикристальной памяти вместо номера прерывания. Аппаратный сброс ($\overline{\text{RS}}$ является активным низким) очищает все флаги фиксирующего прерывания. Заметьте, что если прерывание происходит в то время, как устройство находится в HOLD и HM = 0, когда $\overline{\text{IACK}}$ идет активным низким, адрес присутствовать не будет.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет отображаемый в памяти регистр маски прерывания (IMR) для маскирования внешних и внутренних прерываний. Размещается регистр следующим образом:

15-12	11	10 9	8	7	6	5	4	3	2	1	0
Reserved	HINT	Reserved	$\overline{\text{INT4}}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{\text{INT3}}$	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$

1 в положениях разряда от 15 до 0 регистра IMR разрешает соответствующее прерывание при условии, что INTM = 0. IMR доступен как для операций чтения, так и для операций записи. Когда считывается IMR, любые неиспользованные разряды читаются как один, поскольку все неиспользованные прерывания останавливаются внутри кристалла. Обратите внимание, что ни $\overline{\text{NMI}}$, ни $\overline{\text{RS}}$ не включены в IMR и, следовательно, IMR не воздействует на немаскируемое прерывание или сброс.

Разряд INTM (общее включение), который является 9-ым разрядом регистра состояния ST0, разрешает и отменяет все прерывания. INTM = 0 разрешает все немаскированные прерывания, а INTM = 1 отменяет эти прерывания. INTM устанавливается в 1 автоматически, когда принимается внутреннее прерывание. Если сервисная подпрограмма прерывания выводится, используя инструкцию RETE (возврат из прерывания с автоматическим повторным включением), затем разряд INTM еще раз включается (устанавливается в нуль). Он также может быть установлен в 1 при помощи аппаратного сброса ($\overline{\text{RS}}$ - низкий) или выполняя инструкцию отмены прерывания (SETC INTM). Этот бит сбрасывается в нуль посредством выполнения инструкции разрешения прерывания (CLRC INTM). Обратите внимание, что INTM в данный момент не изменяет IMR или IFR.

Время ожидания прерывания 1867ВЦ2Т, 1867ВЦ2АТ зависит от текущего содержимого конвейера. Устройство всегда заканчивает все инструкции в конвейере перед выполнением программно-управляемого вектора. Когда происходит прерывание, программно-управляемый вектор выбирается на следующий цикл, но три инструкции, в настоящий момент находящиеся на конвейере, также должны быть закончены. Минимальное время ожидания определяется в три цикла, чтобы синхронизировать доходящее прерывание, более трех циклов, чтобы освободить конвейер, и более четырех циклов для выполнения перехода на программно-управляемый вектор. В этой точке выбирается первая инструкция сервисной подпрограммы прерывания (ISR). Следовательно, минимальное время ожидания прерывания составляет десять циклов до начала ISR.

Многоцикловые инструкции добавляют дополнительные циклы, чтобы освободить конвейер. Это применимо к инструкциям, которые растягиваются путем вставки состояний ожидания при обращениях к памяти. Состояния ожидания, требуемые в месте расположения векторов, также воздействуют на время ожидания. N-тактов инструкции повтора (RPT и RPTZ) также блокируют прерывания, и повторяемая инструкция завершает все выполнения перед разрешением обработки прерывания. Это защищает содержимое

повторяемых инструкций, поскольку при повторе инструкции выполняют еще параллельные операции в конвейере, и контекст этих дополнительных параллельных операций не может быть сохранен в $\overline{\text{ISR}}$. Функция HOLD имеет приоритет операций над прерываниями и также может задерживать внутреннее прерывание. Если прерывание происходит во время активного состояния $\overline{\text{HOLD}}$, прерывание будет иметь место по окончании состояния $\overline{\text{HOLD}}$.

Прерывания в последовательности программ не могут выполняться между EINT и следующей инструкцией. Например, если прерывание происходит во время выполнения команды EINT, устройство всегда закончит EINT так же, как и следующую инструкцию перед выполнением фиксирующего прерывания. Это обеспечивает то, что возврат (RET) может быть выполнен перед обработкой следующего прерывания, защищая таким образом от переполнения стек PC. Если ISR выходит, используя RETE (возврат из ISR с разрешением), EINT не является необходимой.

4.1.8.3 Сохранение контекста прерывания

Когда выполняется внутреннее прерывание, автоматически сохраняется определенный стратегический регистр. Когда выполняется возврат из команды прерывания (RETE или RETI), эти регистры автоматически восстанавливаются. Счетчик программ (PC) сохраняется в 8-уровневом аппаратном стеке. Этот стек также используется для вызовов подпрограмм. Следовательно, устройство поддерживает вызовы подпрограмм в рамках сервисной программы прерывания (ISR) до тех пор, пока не превысится 8-уровневый стек. Для следующих регистров также имеется одноуровневый стек:

ACC аккумулятор
ACCB буфер аккумулятора
PREG регистр результата
ST0 0 регистр состояния
ST1 1 регистр состояния
PMST регистр состояния режима процессора
TREG0 временный регистр для умножителя
TREG1 временный регистр для счета сдвига
TREG2 временный регистр для тестирования бита
INDX индексный регистр косвенной адресации
ARCR регистр сравнения вспомогательного регистра

Когда действует внутреннее прерывание, все эти регистры помещаются в стек. Значения в регистрах во время внутреннего прерывания еще доступны для ISR, но также защищены в стеке. Стек выталкивается, когда выполняется возврат из прерывания. Эта система позволяет ЦПУ быть использованным, не требуя сохранения и восстановления контекста в ISR. Это есть одноуровневый стек, поэтому встроенные прерывания не поддерживаются аппаратными средствами. В большинстве случаев это не составляет проблемы, поскольку без сохранения и восстановления контекста прерывания могут выполняться последовательно настолько эффективно, что обработка встроенного прерывания будет менее полезно. Если применение не требует встроенных прерываний, они могут обрабатываться, используя программный стек. Программная совместимость с 1867ВМ2 поддерживается при помощи инструкции RET, используемой для возврата из ISR в процессоре 1867ВМ2, не извлекая из стека эти регистры.

В случае, где ISR необходима для изменения значений в этих регистрах (что касается прерываемых кодов), эти регистры могут извлекаться из стека как показано в следующем примере:

```
ISR
   LACC #ISR_RE_ENTER   ;ACC = адрес точки повторного входа.
   PUSH                 ;Вершина стека = точка повторного входа.
   RETI                 ;Выталкивает все стеки.
ISR_RE_ENTER
   .
   .
   .
   RET                  ;Возврат к прерываемым кодам.
```

В примере адрес точки повторного входа в пределах ISR помещается в стек PC. Инструкция RETI выталкивает все стеки, включая стек PC, и продолжает выполнение. В конце ISR выполняется стандартный возврат, поскольку стек уже вытолкнут.

Векторы для прерываний, не связанные с определенными внешними контактами или внутренними периферийными средствами, могут использоваться как программные прерывания. Соответствующие векторы прерывания могут использоваться как программные внутренние прерывания с полным сохранением и восстановлением контекста посредством выполнения инструкции INTR с соответствующим номером прерывания в качестве операнда. Эти внутренние прерывания защищены от других прерываний тем же путем, что и защищает ISR; все прерывания являются глобально маскируемыми через разряд INTM. Эти подпрограммы внутренних прерываний должны

выходить, используя инструкцию RETI или RETE для выполнения восстановления контекста. Например:

```
INTR  15   ;Программное внутреннее прерывание к адресу 01Eh.
```

В этом примере процессор будет внутренним прерыванием к вектору, относительно помещенному в 01Eh.

Примечание – Хотя взятое устройство может не использовать особый вектор, другие версии могут содержать его, используя номера прерываний от 1 до 15. Инструкция INTR может адресовать 15 дополнительных позиций, чтобы избежать использования зарезервированных ячеек.

4.1.8.4 Немаскируемые прерывания

Оперативная память 1867ВЦ2Т, 1867ВЦ2АТ имеет два немаскируемых прерывания, сброс и $\overline{\text{NMI}}$. Сброс уже был описан ранее.

$\overline{\text{NMI}}$ используется как программно-управляемый сброс. Это отличается от стандартного прерывания, т.к. оно не является маскируемым, и оно не вызывает автоматическое сохранение контекста.

Сохранение контекста не вызывается, потому что возможно принимать $\overline{\text{NMI}}$ равномерно в продолжении сервисной программы прерывания, и стеки сохранения контекста составляют один уровень. $\overline{\text{NMI}}$ отличается от сброса, в котором он не воздействует на какой-либо из режимов устройства. $\overline{\text{NMI}}$ задерживается многоцикловыми инструкциями и режимом $\overline{\text{HOLD}}$. Внутреннее прерывание $\overline{\text{NMI}}$ может также инициироваться программным путем, используя инструкцию $\overline{\text{NMI}}$. Эта инструкция помещает PC ячейку внутреннего прерывания $\overline{\text{NMI}}$.

4.2 Система команд

Система команд 1867ВЦ2Т, 1867ВЦ2АТ поддерживает как операции обработки сигналов, так и программы общего назначения, такие как множественная обработка и быстродействующее управление. Система команд является суперсистемой систем команд процессоров М1867ВМ1 и 1867ВМ2 и совместима сверху вниз с программами обоих устройств. Эта глава описывает инструкции языка ассемблера для цифрового сигнального процессора 1867ВЦ2Т (1867ВЦ2АТ).

4.2.1 Режимы адресации памяти

Система команд 1867ВЦ2Т, 1867ВЦ2АТ обеспечивает шесть основных режимов адресации памяти:

- режим прямой адресации,
- режим косвенной адресации,
- режим непосредственной адресации,
- режим адресации, относящийся к регистрам,
- режим адресации отображаемых в памяти регистров,
- режим циклической адресации.

И прямая и косвенная адресации могут использоваться для обращения к памяти данных. Прямая адресация связывает семь разрядов слова команды с девятью разрядами из указателя страницы памяти данных для формирования 16-разрядного адреса памяти данных. Косвенная адресация обращается к памяти данных через один из восьми вспомогательных регистров. При непосредственной адресации данные основываются на части слов(а) инструкции. Допустимы два вида непосредственной адресации: короткий и длинный. В случае короткой непосредственной адресации 8-/9-/13-разрядный операнд включается в слово инструкции. Режим длинной непосредственной адресации использует в качестве своего операнда 16-разрядное слово, следующее за командой. Адресация, относящаяся к регистру, ссылается на инструкции перемещения блока, в которых программа адресации регистра BMAR или память данных, и инструкции параллельного логического устройства (PLU), в которых операнды достаются из регистра DBMR. Отображаемые в памяти регистры загружаются и хранятся, используя режим адресации отображаемых в памяти регистров.

Циклическая адресация представляет собой дополнительный режим косвенной адресации, который автоматически возвращается к началу блока данных, после того как доходит до конца блока. Следующие подразделы описывают каждый режим адресации, дают форматы кодов операций и несколько примеров для каждого режима.

4.2.1.1 Режим прямой адресации

В режиме прямой адресации памяти инструкция содержит семь младших разрядов из адреса памяти данных (dma). Это поле связывается с семью разрядами регистра указателя страницы (DP) памяти данных для формирования полного 16-разрядного адреса памяти данных. Таким образом, регистр DP указывает на одну из 512 возможных страниц памяти данных, состоящих из 128 слов, а 7-разрядный адрес в инструкции указывает на

определенную ячейку в этой странице памяти данных. Регистр DP загружается, используя инструкции LDP (загрузить указатель страницы памяти данных) или LST #0 (загрузить регистр состояния ST0).

Примечание – Указатель страницы данных не инициализируется сбросом и, следовательно, не определяется подача питания. Пакеты разработки 1867ВЦ2Т, 1867ВЦ2АТ), однако, применяют значения для многих параметров по умолчанию, включая указатель страницы данных. Поэтому программы, которые явно не инициализируют указатель страницы данных, могут выполняться неправильно, в зависимости от того выполняются ли они на устройстве 1867ВЦ2Т (1867ВЦ2АТ) или используя пакеты разработки. Таким образом, принципиально, что все программы инициализируют указатель страницы данных в программном обеспечении.

Рисунок 4.18 показывает, как формируется 16-разрядный адрес данных.

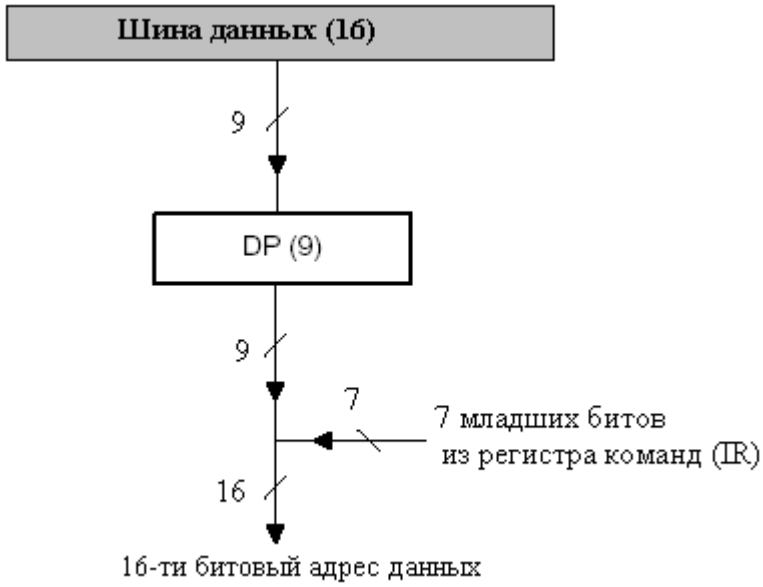


Рисунок 4.18 – Блок-диаграмма прямой адресации

Формат прямой адресации определяется следующим образом:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
код операции								0	dma						

Разряды с 15 по 8 содержат код операции. Разряд 7 = 0 определяет режим адресации как прямой, а разряды с 6 по 0 содержат адрес памяти данных (dma).

Пример формата прямой адресации:

```
ADD  9h,5     ;Содержимое адреса данных 9h сдвигается влево на 5 разрядов и
              ;прибавляется к содержимому аккумулятора.
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1

Код операции инструкции ADD 9h,5 составляет 25h и содержится в разрядах с 15 по 8. Счет сдвига содержится с 11 по 8 разряд кода операции. Адрес памяти данных 09h содержится с 6 по 0 разряд.

4.2.1.2 Режим косвенной адресации

Вспомогательные регистры (AR) обеспечивают гибкую и мощную косвенную адресацию. В 1867ВЦ2Т (1867ВЦ2АТ) имеется в наличии восемь вспомогательных регистров (AR0 – AR7). Для выбора определенного вспомогательного регистра загружается указатель вспомогательного регистра со значением от 0 до 7, определяющий с AR0 по AR7, соответственно (см. рисунок 4.19).

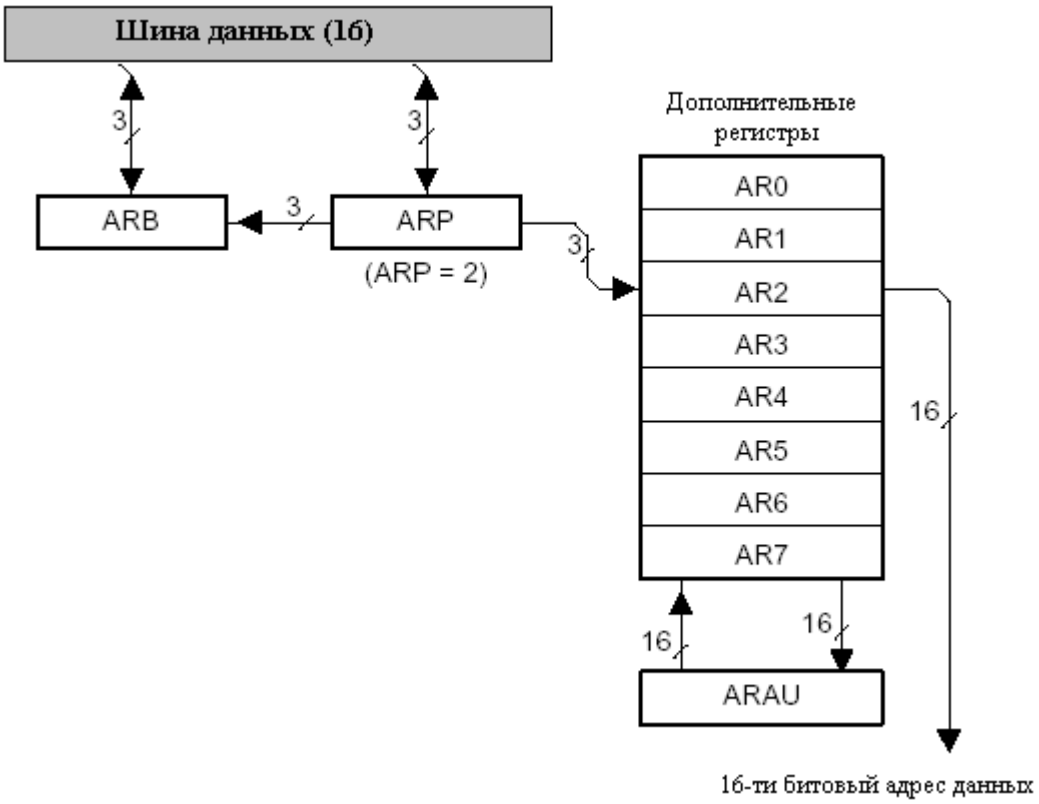


Рисунок 4.19 – Блок-диаграмма косвенной адресации

Текущее содержимое AR используется как адреса операнда памяти данных. После использования инструкции, значение данных текущего AR может быть инкрементировано или декрементировано ARAU, которое реализует беззнаковую 16-разрядную арифметику. ARAU выполняет арифметические операции вспомогательного регистра в стадии декодирования конвейера. Это позволяет адресу формироваться перед фазой декодирования следующей инструкции. Обратите внимание, что увеличение или уменьшение указанного AR эффективно выполняется после использования этого AR в текущей инструкции.

В косвенной адресации любая ячейка в 64К пространстве памяти данных может обращаться через 16-разрядный адрес, содержащийся во вспомогательном регистре. Эти данные могут быть загружены посредством инструкции LAR. Вспомогательные регистры процессора 1867ВЦ2Т (1867ВЦ2АТ) могут быть модифицированы посредством ADRK (прибавить к вспомогательному регистру короткий непосредственный) или SBRK (вычесть из вспомогательного регистра короткий непосредственный). Вспомогательные регистры процессора 1867ВЦ2Т (1867ВЦ2АТ) также могут быть модифицированы инструкцией MAR (модифицировать вспомогательный регистр) или, равносильно, полем косвенной адресации любой инструкции, поддерживающей косвенную адресацию. AR(ARP) обозначает, что вспомогательный регистр выбирается посредством ARP. Вспомогательные регистры могут также загружаться через шину данных, используя отображаемые в памяти записи во вспомогательных регистрах. Следующие инструкции могут производить запись в отображаемые в памяти вспомогательные регистры: APLK, BLDD, BLDP, LMMR, OPLK, SACH, SACL, SAMM, SMMR, SPLK и XPLK. Пользователь должен быть внимателен, используя эти отображаемые в памяти загрузки вспомогательных регистров и инструкцию либо SBRK, либо ADRK. В обоих случаях отображаемые в памяти вспомогательные регистры модифицируются в стадии выполнения конвейера. Это будет вызывать конфликт конвейера, если одно из следующих двух слов инструкции модифицирует этот вспомогательный регистр.

Приведенные ниже символы используются при косвенной адресации, включая бит-реверсивную (BR) адресацию:

*	Содержание AR(ARP) используется как адрес памяти данных.
*-	Содержание AR(ARP) используется как адрес памяти данных, и оно уменьшается на 1 после доступа.
*+	Содержание AR(ARP) используется как адрес памяти данных, и оно увеличивается на 1 после доступа.
*0-	Содержание AR(ARP) используется как адрес памяти данных, и из него вычитается содержание INDX после доступа.
*0+	Содержание AR(ARP) используется как адрес памяти данных, и к нему добавляется содержание INDX после доступа.
*BR0-	Содержание AR(ARP) используется как адрес памяти данных, и из него вычитается содержание INDX с реверсивным переносом (rc) наследования от этого после доступа.
*BR0+	Содержание AR(ARP) используется как адрес памяти данных, и к нему добавляется содержание INDX с реверсивным переносом (rc) наследования от этого после доступа.

Существует два основных типа косвенной адресации с индексированием:

- регулярная косвенная адресация с инкрементом или декрементом,
- регулярная косвенная адресация с индексированием основанном на значении INDX:
  - индексирование путем добавления или вычитания содержания INDX, или
  - индексирование путем добавления или вычитания содержания INDX с реверсивным переносом наследования.

В любом случае содержание дополнительного регистра, на который указывает регистр ARP используется как адрес операнда памяти данных. Потом ARAU выполняет специальную математическую операцию над указанным дополнительным регистром. Вдобавок, ARP может быть загружен с новым значением. Все индексированные операции выполняются в текущем дополнительном регистре в том же цикле, что и специальная инструкция фазы декодирования на конвейере.

Косвенная адресация дополнительного регистра позволяет регулировать пост-доступ для дополнительного регистра, на который указывает ARP. Регулировка может быть инкрементной или декрементной на содержимое INDX регистра. Совместимость с процессором с 1867ВМ2 осуществляется путем обнуления NDX бита в PMST регистре. В архитектуре с 1867ВМ2 инкрементирование или декрементирование текущего дополнительного регистра может быть сделано благодаря использованию значения регистра AR0. При обнуление NDX бита, каждое значение, загружаемое в AR0, будет автоматически загружаться и в INDX регистр. Последующие изменения текущего дополнительного регистра на значение отличное от 0 будут использовать регистр INDX, чем достигается совместимость с 1867ВМ2. При перезагрузке NDX бит обнуляется.

Бит-реверсный адресный режим на 1867ВЦ2Т, 1867ВЦ2АТ позволяет эффективно исполнять алгоритмы быстрого преобразования Фурье (БПФ). Направление переноса наследования в ARAU реверсивное, когда выбран этот режим, и INDX добавляется/вычитается к/из текущего дополнительного регистра. Типичное использование этого адресного режима требует, чтобы в INDX было загружено значение, определяющее размер массива, и в AR(ARP) был загружен базовый адрес данных (первая ссылка на данные).

В косвенном режиме адресации можно использовать все команды за исключением команд с непосредственными операндами или без операндов. Формат косвенной адресации следующий:

15 14 13 12 11 10 9 8	7	6	5	4	3	2 1 0
Opcode	1	IDV	INC	DEC	NAR	Y

Биты с 15 по 8 содержат код операции, бит 7 = 1 указывает на то, что установлен косвенный режим адресации. Биты с 6 по 0 содержат биты контроля косвенной адресации.

Бит 6 содержит значение инкремента/декремента (IDV). Бит IDV определяет, когда INDX регистр будет использован для инкрементации или декрементации текущего дополнительного регистра. Если бит 6 = 0, то происходит инкрементирование или декрементирование текущего дополнительного регистра. Если бит 6 = 1, то INDX регистр добавляется или вычитается из текущего дополнительного регистра, определенного 5 и 6 битами.

Биты 5 и 4 контролируют арифметические операции, которые выполняются с AR(ARP) и INDX регистрами. Если установлен 5 бит, то будет выполнено инкрементирование. Если установлен 4 бит, то будет выполнено декриментирование. Таблица 4.8 показывает соответствие значения бита и арифметической операции.

Таблица 4.8 – Арифметические операции косвенной адресации

Биты			Арифметические операции
6	5	4	
0	0	0	Нет операций
0	0	1	AR(ARP) – 1 → AR(ARP)
0	1	0	AR(ARP) + 1 → AR(ARP)
0	1	1	Зарезервировано
1	0	0	AR(ARP) – INDX → AR(ARP) [реверсивный перенос наследования]
1	0	1	AR(ARP) – INDX → AR(ARP)
1	1	0	AR(ARP) + INDX → AR(ARP)
1	1	1	AR(ARP) + INDX → AR(ARP) [реверсивный перенос наследования]

Бит 3 и биты со 2 по 1 контролируют указатель на дополнительный регистр (ARP). Бит 3 (NAR) установлен, если новое значение загружено в ARP. Если бит 3 = 1, то содержание битов со 2 по 0 (Y = следующему ARP) загружается в ARP.

Если бит 3 = 1, то содержание ARP не изменяется. Если в ARP загружается новое значение, то старое значение загружается из буфера дополнительного регистра (ARB) в ST1 регистр статуса.

Таблица 4.9 показывает битовые поля, написание и операторы, используемые для косвенной адресации.

Таблица 4.9 – Битовые поля при косвенной адресации

Биты командного поля 15 – 8 7 6 5 4 3 2 1 0	Написание	Операция
1	2	3
<-Opcode->1 0 0 0 0 < -Y->	*	Нет изменений в Arx/ARP
<-Opcode->1 0 0 0 1 < -Y->	*,Y	Y→AR(ARP)

Продолжение таблицы 4.9

1	2	3
<-Opcode->1 0 0 1 0 < -Y->	*-	AR(ARP) – 1 → AR(ARP)
<-Opcode->1 0 0 1 1 < -Y->	*-,Y	AR(ARP) – 1 → AR(ARP) Y→ARP
<-Opcode->1 0 1 0 0 < -Y->	*+	AR(ARP) + 1 → AR(ARP)
<-Opcode->1 0 1 0 1 < -Y->	*+,Y	AR(ARP) – 1 → AR(ARP) Y→ARP
<-Opcode->1 1 0 0 0 < -Y->	*BRO-	AR(ARP)-rcINDX → AR(ARP) @
<-Opcode->1 1 0 0 1 < -Y->	*BRO-,Y	AR(ARP)-rcINDX → AR(ARP) Y→ARP @
<-Opcode->1 1 0 1 0 < -Y->	*0-	AR(ARP) – INDX → AR(ARP)
<-Opcode->1 1 0 1 1 < -Y->	*0-,Y	AR(ARP) – INDX → AR(ARP) Y→ARP
<-Opcode->1 1 1 0 0 < -Y->	*0+	AR(ARP) + INDX → AR(ARP)
<-Opcode->1 1 1 0 1 < -Y->	*0+,Y	AR(ARP) + INDX → AR(ARP) Y→ARP
<-Opcode->1 1 1 1 0 < -Y->	*BRO+	AR(ARP) + rcINDX → AR(ARP) @
<-Opcode->1 1 1 1 1 < -Y->	*BRO+,Y	AR(ARP) + rcINDX → AR(ARP) Y→ARP @
Примечание - @ BR = бит-реверсный режим адресации и rc = реверсный перенос наследования.		

CMPR (сравнение дополнительного регистра с ARCR) и условия TC/NTC облегчают условные ветвления, вызовы, возвращения или условное выполнение, базирующееся на сравнении содержания ARAR с содержанием AR(ARP). Совместимость с устройствами TMS320C2x поддерживается путем обнуления NDX бита в PMST регистре. В архитектуре TMS320C2x функция сравнения дополнительного регистра выполняется путем сравнения AR0 с текущим дополнительным регистром. Благодаря обнулению NDX бита при каждой загрузке нового значения в AR0 такое же значение будет загружено в регистр ARCR. При перезагрузке NDX бит обнуляется. Дополнительный регистр может также быть использован для временного хранения в командах загрузки и хранения дополнительного регистра, предпочтительно с помощью LAR и SAR или при любых других операциях загрузки и хранения отображаемого в память дополнительного регистра.

Следующие примеры иллюстрируют косвенную адресацию:

```
1. ADD *+, 8
```

Прибавляет к аккумулятору содержимое памяти данных по адресу, определенному содержимым текущего дополнительного регистра. Данные перед прибавлением сдвигаются влево на 8 бит. Текущий дополнительный регистр увеличивается на 1. Код инструкции 28F0h.

```
2. ADD *, 8
```

То же что в примере 1, но без автоинкремента; код команды 2880h.

```
3.  ADD *-, 8
```
То же что в примере 1, но с автодекрементом; код команды 2890h.

```
4.  ADD *0+, 8
```

То же что в примере 1, но содержимое регистра INDX прибавляется к текущему дополнительному регистру; код команды 2890h.

```
5.  ADD *0-, 8
```

То же что в примере 4, но содержимое регистра INDX вычитается из текущего дополнительного регистра; код команды 28D0h.

```
6.  ADD *+, 8, AR3
```

То же что в примере 1, но в указатель дополнительного регистра (ARP) загружается значение 3 для последующих инструкций; код команды 28Abh.

```
7.  ADD *BR0-, 8
```

Содержимое регистра INDX вычитается из текущего дополнительного регистра с изменением порядка бит (reverse carry propogation); код команды 28C0h.

```
1000  ADD *BR0+, 8
```

Содержимое регистра INDX прибавляется к текущему дополнительному регистру с изменением порядка бит (reverse carry propogation); код команды 28F0h.

4.2.1.3 Режим непосредственной адресации

При непосредственной адресации слово(а) инструкции содержит(ат) значение непосредственного операнда. Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет однословные (8-, 9-, 13-битные) короткие непосредственные инструкции и двухсловные (16 бит) длинные непосредственные инструкции. Для коротких непосредственных инструкций непосредственный операнд содержится в слове инструкции. Для длинных непосредственных инструкций в качестве непосредственного операнда используется следующее слово инструкции.

Непосредственную адресацию поддерживают следующие инструкции:

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

Пример короткого непосредственного формата:

RPT #99 ; выполнить следующую за RPT инструкцию 100 раз.

В RPT непосредственный операнд содержится в части слова инструкции:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	8-bit константа							

Для длинных непосредственных инструкций в качестве непосредственного операнда используется следующее слово инструкции. 16-битное значение может использоваться как абсолютное значение или отрицательное число (дополнение до 2).

RPT #0fffh ; выполнить следующую за RPT инструкцию 4096 раз

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	0	0
16-битная константа															

4.2.1.4 Адресация выделенного регистра

Девять инструкций 1867ВЦ2Т, 1867ВЦ2АТ могут использовать один из двух специальных отображаемых в память регистров – (BMAR) – регистр адреса перемещения блока и DBMR (Dinamic Bit Manipulation Register). Инструкции APL, OPL, CPL и XPL устройства параллельной логики (PLU) используют содержимое DMBR, если не определен один из операндов. Инструкции BLDD, BLDP и PLPD могут использовать DMBR для указания на источник или приемник при перемещении блока. MADD и MADS используют BMAR для адресации операнда при умножении в памяти программ.

Такой способ адресации используется в случае явного указывания регистра BMAR в команде или исключения непосредственного операнда из операции PLU. Использование BMAR подразумевается в MADD и MADS.

```
BLLD  BMAR, DAT100 ; DP = 0, BMAR содержит значение 200h
```

Содержимое ячейки памяти данных 200h копируется в память данных по адресу 100h в текущей странице. Код операции 0ac64h.

```
OPL   DAT10          ; DP = 6 , DMBR содержит 0fff0h
                       ; ячейка 30ah содержит 1
```

Содержимое памяти данных по адресу 30ah логически складывается с содержимым DMBR. Результат 0fff1h записывается обратно в ячейку 30ah. Код операции 590ah.

4.2.1.5 Адресация отображаемых в память регистров

Адресация отображаемых в память регистров используется для модификации отображаемых в память регистров без изменения значения текущего указателя страницы. В этом режиме может быть модифицирована ячейка памяти в странице данных 0. Для этого 9 старших битов адреса обнуляется независимо от текущего значения DP или дополнительного регистра. Значение DP не меняется.

Рисунок 4.20 – Адресация регистров, отображаемых в память

Этот режим адресации обеспечивает гибкость в работе с отображаемыми в память регистрами. Издержки при использовании отображаемых в память регистров невелики, т.к. указатель страницы (DP) не модифицируется после операции. Адресация отображаемых в память регистров выполняется следующими инструкциями:

- LAMM – загрузить отображаемый в память регистр в аккумулятор,
- SAMM – записать аккумулятор в отображаемый в память регистр,
- LMMR – загрузить отображаемый в память регистр,
- SMMR – записать отображаемый в память регистр.

Примеры 1 и 2 показывают использование этих инструкций в прямом и косвенном режиме адресации.

```
LMMR CBCR, #0800h   ; DP=6, загрузка регистра CBCR
```

*В отображаемый в память регистр **CBCR** загружается значение **800h**. Код операции **590ah** и **0800h** во втором слове.*

```
SAMM   *+           ; запись ACC в регистр PMST
```

Если указатель дополнительного регистра ARP=3 и дополнительный регистр AR3 = 0ff07h, содержимое аккумулятора записывается в регистр PMST (адрес 7h), указанный младшими 7 битами AR3. Код операции 08890h.

4.2.1.6 Циклическая адресация

Многие алгоритмы, такие как свертки, корреляции и КИХ фильтры могут быть реализованы с использованием циклических буферов в памяти. В этих алгоритмах циклический буфер может использоваться для создания окна, содержащего обрабатываемые данные. 1867ВЦ2Т (1867ВЦ2АТ) поддерживает 2 циклических буфера, использующих дополнительные регистры. Следующие 5 отображаемых в память регистров управляют операциями с циклическими буферами:

- CBSR1 – регистр начала циклического буфера 1,
- CBSR2 – регистр начала циклического буфера 2,
- CBER1 – регистр конца циклического буфера 1,
- CBER2 – регистр конца циклического буфера 2,
- CBCR - регистр управления циклического буфера.

8 – битовый регистр управления циклического буфера разрешает или запрещает операции с циклическим буфером.

Отдельные поля CBCR:

Бит	Название	Функция
0-2	CAR1	Определяет дополнительный регистр, связанный с циклическим буфером 1
3	CENB1	Циклический буфер 1 разрешен=1/запрещен=0. Устанавливается на 0 при сбросе.
4-6	CAR2	Определяет дополнительный регистр, связанный с циклическим буфером 2
7	CENB2	Циклический буфер 2 разрешен=1/запрещен=0. Устанавливается на 0 при сбросе.

Чтобы определить циклический буфер, сначала загружается начальный и конечный адрес буфера в соответствующие регистры; затем значение, находящееся между начальным и конечным адресом, загружается в дополнительный регистр. Затем устанавливается номер индексного регистра и бит разрешения циклического буфера в CBCR. Алгоритм циклической адресации :

```
if(Arn == CBER)        ; проверка ДО модификации !
    Arn = CBSR;
else
    Arn = Arn + step;
```

step – значение, которое будет прибавлено или вычтено из заданного дополнительного регистра. Необходимо соблюдать осторожность в случаях когда step > 1. Если адрес выйдет за пределы циклического буфера, ARAU не распознает эту ситуацию и буфер не вернётся на начало.

Может использоваться инкрементный или декрементный способ адресации циклических буферов. При инкрементной адресации значение CBER > CBCR, при декрементной – CBER < CBCR.

4.2.2 Система команд

В следующем подразделе перечислены символы и аббревиатуры, используемые в обзоре набора команд и в описании инструкций. Обзор набора команд сгруппирован в соответствии с их функциями. Подробное описание каждой инструкции приведено далее.

4.2.2.1 Обозначения и символы

Таблица 4.10 содержит список символов и аббревиатур.

Таблица 4.10 - Список символов и аббревиатур

Символ	Значение
1	2
A	Адрес
ACC	Аккумулятор
ARB	Буфер указателя дополнительного регистра
ARn, Arx	Дополнительный регистр n($0 \leq n \leq 7$)
ARP	Указатель дополнительного регистра
BIO	Вход управления переходом
BMAR	Регистр адреса перемещения блока
C	Бит переноса
CM	2-битовое поле, определяющее режим сравнения
CNF	Бит управления конфигурацией внутренней ДОЗУ
DATn	Метка, присвоенная ячейке памяти данных n
DBMR	Регистр управления динамическим битом
dma	Адрес памяти данных
DP	Указатель страницы данных
FSM	Режима синхронизации фреймов
HM	Бит режима hold
I	Бит режима адреса
ind	Косвенный адресный операнд
INTM	Флаг режима прерываний
K	Поле непосредственного операнда
MCS	Стек микровызова

Продолжение таблицы 4.10

1	2
nnh	nn в шестнадцатеричном представлении
OV	Бит переполнения
OVM	Бит режима переполнения
P	Регистр результата
PC	Счетчик команд
PFC	Предварительный счетчик
PGMn	Метка, присвоенная ячейке памяти программ n
PM	2-битное поле, определяющее режим сдвига выхода регистра P
pma	Адрес памяти программ
RPTC	Счетчик повторов цикла.
S	4-битный код сдвига влево
STn	Регистр состояния n (n = 0 или 1)
SXM	Бит режима расширения знака
TREGn	Временный (вспомогательный) регистр n ($0 \leq n \leq 2$)
TC	Бит тест-контроля
TOS	Вершина стека
TRM	Управляющий бит совместимости по TREG-регистрам
TXM	Бит режима передачи/переноса
XF	Статусный бит XF
$\rightarrow$	Присвоение
| x |	Абсолютное значение
[ ]	Необязательные элементы
( )	Содержимое
{ }	Альтернативные элементы (один должен присутствовать)
#	Префикс констант, используемых в непосредственной адресации

4.2.2.2 Краткое изложение системы команд

Таблица 4.11 – это обзор команд процессора 1867ВЦ2Т (1867ВЦ2АТ).

Дополнительная информация находится в индивидуальном описании инструкций в следующем подразделе.

Таблица 4.11 – Сводка инструкций (команд)

Инструкции, работающие с аккумулятором		
Мнемоника	Описание	Слов
1	2	3
ABS	Абсолютное значение ACC	1
ADCB	Сложить ACCB и бит переноса с ACC	1
ADD	Прибавить данные к ACC	1/2
ADDB	Прибавить ACCB к ACC	1
ADDC	Прибавить данные и бит переноса к ACC	1
ADDS	Прибавить данные к ACC с подавлением расширения знака	1
ADDT	Прибавить данные к ACC со сдвигом, заданным в TREG1	1
AND	Данные AND с ACC	1/2
ANDB	ACCB AND ACC	1
BSAR	Барабанный сдвиг ACC вправо	1
CMPL	Дополнить до 2 ACC	1
CRGT	Проверка на ACC > ACCB	1
CRLT	Проверка на ACC < ACCB	1
EXAR	Поменять значениями ACC и ACCB	1
LACB	Загрузить ACCB в ACC	1
LACC	Загрузить данные в ACC со сдвигом	1/2
LACL	Загрузить данные в младшее слово ACC	1
LACT	Загрузить данные в ACC со сдвигом, заданным в TREG1	1
LAMM	Загрузить в ACC содержимое отображаемого в памяти регистра	1
NEG	Изменить знак ACC	1
NORM	Нормализация ACC	1
OR	Данные OR с ACC	1/2
ORB	ACCB OR ACC	1/2
ROL	Циклический сдвиг ACC влево	1
ROLB	Циклический сдвиг ACCB и ACC влево	1
ROR	Циклический сдвиг ACC вправо	1
RORB	Циклический сдвиг ACCB и ACC вправо	1
SACB	Сохранить ACC в ACCB	1
SACH	Сохранить старшее слово ACC со сдвигом	1
SACL	Сохранить младшее слово ACC со сдвигом	1
SAMM	Сохранить ACC в отображаемый в памяти регистр	1
SATH	Барабанный сдвиг вправо на 0-16 бит ACC как задано регистром TREG1	1
SATL	Барабанный сдвиг ACC вправо как задано регистром TREG1	1
SBB	Вычесть ACCB из ACC	1
SBBB	Вычесть ACCB и бит переноса из ACC	1
SFL	Сдвиг ACC влево	1
SFLB	Сдвиг ACC и ACCB влево	1
SFR	Сдвиг ACC вправо	1
SFRB	Сдвиг ACC и ACCB вправо	1
SUB	Вычесть данные из ACC	1/2
SUBB	Вычесть данные и бит переноса из ACC	1
SUBC	Условное вычитание	1
SUBS	Вычесть данные из ACC с подавлением расширения знака	1
SUBT	Вычесть данные из ACC со сдвигом, заданным в TREG1	1
XOR	Данные XOR ACC	1/2
XORB	ACCB XOR ACC	1
ZALP	Обнулить младшее слово ACC и загрузить старшее; с округлением	1
ZAP	Обнулить ACC и PREG	1

Продолжение таблицы 4.11

Инструкции, работающие с дополнительными регистрами и указателем страниц		
Мнемоника	Описание	Слов
ADRK	Прибавить к ARn короткий непосредственный операнд	1
CMPR	Сравнить ARn с ARCR	1
LAR	Загрузить ARn	1/2
LDP	Загрузить указатель страницы данных	1
MAR	Модифицировать ARn	1
SAR	Сохранить ARn	1
SBRK	Вычесть из ARn короткий непосредственный операнд	1
Инструкции PLU		
Мнемоника	Описание	Слов
APL	Данные AND DMBR/константой	1/2
CPL	Сравнить DMBR/константу с данными	1/2
OPL	Данные OR DMBR/константу	1/2
SPLK	Записать длинную непосредственную константу в память данных	2
XPL	Данные XOR DMBR/константу	1/2
T-регистр, P-регистр и операции умножения		
Мнемоника	Описание	Слов
APAC	Прибавить PREG к ACC	1
LPH	Загрузить данны в старшее слово PREG	1
LT	Загрузить данные в TREG0	1
LTA	Загрузить данные в TREG0 и добавить PREG к ACC	1
LTD	Загрузить данные в TREG0 и добавить PREG к ACC; переместить данные	1
LTP	Загрузить данные в TREG0 и сохранить PREG в ACC	1
LTS	Загрузить данные в TREG0 и вычесть PREG из ACC	1
MAC	Умножить с накоплением	2
MACD	Умножить с накоплением; переместить данные	2
MADD	Умножить с накоплением на значение BMAR; переместить данные	1
MADS	Умножить с накоплением на значение BMAR	1
MPY	Умножить	1/2
MPYA	Умножить с накоплением	1
MPYS	Умножить и вычетанием	1
MPYU	Беззнаковое умножение	1
PAC	Сохранить PREG в ACC	1
SPAC	Вычесть PREG из ACC	1
SPH	Сохранить старшее слово PREG со сдвигом определяемым битом PM	1
SPL	Сохранить младшее слово PREG со сдвигом определяемым битом PM	1
SPM	Задать сдвиг PREG регистра	1
SQRA	Возвести в квадрат с накоплением	1
SQRS	Возвести в квадрат с вычетанием	1
ZPR	Обнулить PREG	1
Инструкции перехода		
Мнемоника	Описание	Слов
B[D]	Безусловный переход	2
BACC[D]	Переход по адресу из ACC	1
BANZ[D]	Переход по ARn != 0	2
BCND[D]	Условный переход	2
CALA[D]	Вызов подпрограммы по адресу из ACC	1
CALL[D]	Вызов подпрограммы	2
CC[D]	Условный вызов подпрограммы	2
INTR	Программное прерывание	1
NMI	Немаскируемое прерывание	1
RET	Возврат из подпрограммы	1

Продолжение таблицы 4.11

1	2	3
RETC[D]	Условный возврат из подпрограммы	1
RETE	Возврат с переключением контекста и глобальным разрешением прерываний	1
RETI	Возврат с переключением контекста	1
TRAP	Программное прерывание	1
XC	Условное выполнение следующей инструкции	1
Операции в/из операции с памятью		
Мнемоника	Описание	Слов
BLDD	Перемещение блока из памяти данных в память данных	1/2
BLDP	Перемещение блока из памяти данных в память программ	1
BLPD	Перемещение блока из памяти программ в память данных	1/2
DMOV	Перемещение данных в памяти данных	1
IN	Ввод данных из порта	2
LMMR	Чтение отображаемого в памяти регистра	2
OUT	Вывод данных в порт	2
SMMR	Запись отображаемого в памяти регистра	2
TBLR	Чтение таблицы данных	1
TBLW	Запись таблицы данных	1
Команды управления		
Мнемоника	Описание	Слов
BIT	Тестирование бита	1
BITT	Тестирование бита, определенного TREG2	1
CLRC	Очистка бита управления	1
IDLE	Ожидание прерывания	1
IDLE2	Ожидание прерывания - режим малого потребления энергии	1
LST	Загрузить регистр статуса	1
NOP	Нет операции	1
POP	Вытолкнуть вершину стека в младшее слово ACC	1
POPD	Вытолкнуть вершину стека в память данных	1
PUSHD	Поместить значение из памяти данных в стек	1
PUSH	Поместить младшее слово ACC в стек	1
RPT	Повтор следующей инструкции	1/2
RPTB	Повтор блока инструкций	2
RPTZ	Повтор следующей инструкции и очистка ACC и PREG	2
SETC	Установить бит управления	1
SST	Сохранить регистр статуса	1

4.2.3 Описание команд

Этот подраздел дает подробное описание командам из таблицы 4.11. Каждая команда представляет следующую информацию:

- синтаксис ассемблера,
- операнды,
- исполнение,
- описание,
- слова,
- циклы,
- примеры.

EXAMPLE

```
СИНТАКСИС  Прямая:                    EXAMPLE dma[,сдвиг]
           Косвенная:                 EXAMPLE {индекс}[,сдвиг[,новый ARP]]
           Короткая непосредсвенная: EXAMPLE [#k]
           Длинная  непосредсвенная: EXAMPLE [#lk]
```

Каждая команда записывается в синтаксисе ассемблера. Необязательное поле комментария может заключать синтаксическое выражение. Между каждым полем (команда, операнд, комментарии) требуется наличие пробела.

```
ОПЕРАНДЫ        0 ≤ dma ≤ 127
                0 ≤ pma ≤ 65535
                0 ≤ новый ARP ≤ 7
                0 ≤ k ≤ 255
                0 ≤ lk ≤ 65535
                0 ≤ сдвиг ≤ 15
                индексы: {*|*+|*-|*0+|*0-|*BRO-|*BRO+}
```

Представленный выше список операндов не полный, однако, он содержит наиболее часто используемые. Операнды могут быть константами или временными ассемблерными выражениями, обращающиеся к памяти, портам ввода/вывода, адресным регистрам, ссылкам, сдвигающим счетчикам и другим константам.

```
ИСПОЛНЕНИЕ      (PC) + 1 -> PC
                (ACC) + (dma) -> ACC;
                0 -> C
                Зависит от OVM; изменяет OV и C. Не зависит от SXM.
```

Команды операции последовательно описывают процесс выполнения команды. Также даны условные эффекты, зависящие от заданных статусным регистром режимов. Также описаны те биты в статусных регистрах 1867ВЦ2Т(1867ВЦ2АТ), на которые влияет данная команда.

```
ОПИСАНИЕ        Описывается исполнение команд и эффект на ожидае-
                мое содержание процессора или памяти. Описаны
                все ограничения, налагаемые на операнд процессором
                или ассемблером. Описание дублирует и дополняет
                блок исполнения.

СЛОВА           Здесь определяется количество слов памяти тре-
                буемое для хранения команды и ее дополнительных слов.

ЦИКЛЫ           Здесь определяется количество циклов требуемое для выпол-
                нения команды. Все команды исполняются из внутрен-
                ней программной памяти (ROM) используя внутреннюю RAM
                данных с двойным доступом. Количество циклов указыва-
                ется для однократно выполняемой команды. Заметьте,
                что чтение или запись любого размещенного в памяти
                переферийного регистра на TI шине добавляет один
                дополнительный цикл на выполнение этой команды.

ПРИМЕР          Для каждой команды дан пример кода. Представлено влия-
                ние кода на память и/или регистры.
```

ABS - Вычисление абсолютного значения аккумулятора

СИНТАКСИС ABS

ОПЕРАНДЫ нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0

ИСПОЛНЕНИЕ (PC) + 1 -> PC
|(ACC)| -> ACC; 0 -> C
Зависит от OVM; влияет на OV и C. Не зависит от SXM.

ОПИСАНИЕ Если содержание аккумулятора больше или равно нулю, то после исполнения ABS оно не меняется. В противном случае в аккумулятор загружается значение добавленное до двух. После выполнения этой команды в бит переноса (C) всегда устанавливается нуль.

Заметьте, что 80000000h - особый случай. Когда режим переполнения не установлен (OVM = 0) ABS от 80000000h есть 80000000h. Когда режим переполнения установлен (OVM = 1) ABS от 80000000h есть 7FFFFFFFh. В любом случае OV бит устанавливается в 1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
ABS
До выполнения                 После выполнения
ACC=1234h; C=X                ACC=1234h; C=0
```

ПРИМЕР 2
```
ABS
До выполнения                 После выполнения
ACC=FFFFFFFFh; C=X            ACC=1h; C=0
```

ПРИМЕР 3
```
ABS ; (OVM = 1)
До выполнения                 После выполнения
ACC=80000000h; C=X            ACC=7FFFFFFFh; C=0
               OV=X                          OV=1
```

ПРИМЕР 4
```
ABS ; (OVM = 0)
До выполнения                 После выполнения
ACC=80000000h;  C=X           ACC=80000000h;  C=0
                OV=X                          OV=1
```

ADCB – Добавление буферу аккумулятора и бита переноса к аккумулятору

СИНТАКСИС ADCB

ОПЕРАНДЫ нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	0	1

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) + (ACCB) + (C) -> ACC
Зависит от OVM; влияет на OV и C

ОПИСАНИЕ Содержимое буфера аккумулятора (ACCB) и значение бита переноса (C) прибавляется к аккумулятору. Результат сохраняется в аккумуляторе.Бит переноса устанавливается в 1, если результат сложения генерирует перенос; иначе бит C устанавливается 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР ADCB

До выполнения		После выполнения	
ACC	1234h; C=1	ACC	1237h; C=0
ACCB	2h	ACCB	2h

ADD – Добавление со сдвигом к аккумулятору

СИНТАКСИС

Прямая: ADD dma[,сдвиг]
Косвенная: ADD {индекс}[,сдвиг][,новый ARP]]
Короткая непосредственная: ADD #k
Длинная непосредственная: ADD #k[,сдвиг]

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ сдвиг1 ≤ 15 (по умолчанию 0)
0 ≤ новый ARP ≤ 7
0 ≤ k ≤ 255
-32768 ≤ lk ≤ 32767

КОД КОМАНДЫ

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	SHFT†				0	dma						

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	SHFT†				1	See Section 5.2						

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	0	dma						

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0	8-Bit Constant							

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	0	1	SHFT†			
16-Bit Constant															

ИСПОЛНЕНИЕ

Прямая и косвенная адресации:
(PC) + 1 -> PC
(ACC) + [(dma)*$2^{сдвиг}$] -> ACC
Зависит от SXM и OVM; влияет на C и OV.

Короткая непосредственная адресация:
(PC) + 1 -> PC
(ACC) + k -> ACC
Зависит от OVM; влияет на C и OV.

Длинная непосредственная адресация:
(PC) + 2 -> PC
(ACC) + lk*$2^{сдвиг}$ -> ACC
Зависит от OVM и SXM; влияет на C и OV.

ОПИСАНИЕ

Содержание адресуемой ячейки памяти данных или непосредсвенная константа сдвигается влево, в соответствии со значением сдвига, и прибавляется к аккумулятору. При выполнении сдвига младшие разряды прибавляемых данных заполняются нулями. В старшие разряды аккумулятора записывается значение знака, если SXM = 1 или они заполняются нулями, если SXM = 0. Результат запоминается

в аккумуляторе.
При модификации ARP при косвенной адресации необходимо указывать значение сдвига. Если сдвиг не используется, указывается 0: ADD *+,0,AR0
Короткий непосредственный 8-битный операнд добавляется к содержимому ACC как положительное значение. Результат операции сохраняется в ACC. В этом режиме не выполняется операций сдвига и операции контроля знака через SXM бит. C бит устанавливается в 1 если результат операции генерирует перенос; иначе C бит устанавливается в 0. Если в инструкции задается сдвиг на 16 C бит устанавливается только если результат добавления генерирует перенос; иначе бит C не модифицируется. Это позволяет корректно генерировать одиночный перенос при добавлении 32-битного числа к ACC.

СЛОВА 1 (Прямая, косвенная или короткая непосредственная адресация).
2 (Длинная непосредственная адресация).

ЦИКЛЫ

Cycles for a Single Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
ADD DAT1,1  ;(DP = 6)
До выполнения                 После выполнения
Память данных                 Память данных
301h    1h                    301h    1h
ACC     2h;  C=X              ACC     04h;  C=0
```

ПРИМЕР 2
```
ADD *+,0,AR0
До выполнения                 После выполнения
ARP     4                     ARP     0
AR4     0302h                 AR4     0303h
Память данных                 Память данных
302h    2h                    302h    2h
ACC     2h;  C=X              ACC     04h;  C=0
```

```
ПРИМЕР 3        ADD #1h   ; короткий непосредственный операнд
                До выполнения                  После выполнения
                ACC     2h;  C=X               ACC     03h;  C=0

ПРИМЕР 4        ADD #1111h,1   ; длинный непосредственный операнд
                                 со сдвигом 1
                До выполнения                  После выполнения
                ACC     2h; C=X                ACC     2224h; C=0
```

ADDB – Добавление буфера аккумулятора к аккумулятору

СИНТАКСИС ADDB

ОПЕРАНДЫ Нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) + (ACCB) -> ACC
Зависит от OVM; влияет на C и OV.

СЛОВА Содержание буфера аккумулятора (ACCB) прибавляется к аккумулятору.Результат сохраняется в аккумуляторе, а содержимое ACCB не изменяется. Бит переноса устанавливается в 1, если результат генерирует перенос; иначе бит C устанавливается 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT)Instruction

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР ADDB

До выполнения			После выполнения		
ACC	1234h		ACC	1236h	
ACCB	2h;	C=X	ACCB	2h;	C=0

ADDC – Добавление значения памяти данных и бита переноса к аккумулятору

СИНТАКСИС
Прямая: ADDC dma
Косвенная: ADDC {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ
(PC) + 1 -> PC
(ACC) + dma + (C) -> ACC
Не зависит от OVM; влияет на C и OV.

ОПИСАНИЕ
Значение адресуемой ячейки памяти данных и значение бита переноса прибавляется к аккумулятору. Результат сохраняется в аккумуляторе. Бит переноса устанавливается в 1, если результат генерирует перенос; иначе бит C устанавливается 0.
Команда ADDC может быть использована для выполнения арифметических действий высокой точности.

СЛОВА
1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
ADDC DAT0  ; (DP = 6)
До выполнения                 После выполнения
Память данных                 Память данных
300h    04h                   300h    04h
ACC     13h;  C=1              ACC     18h;  C=0
```

ПРИМЕР 2
```
ADDC *-,AR4 ; (OVM = 0)
До выполнения                 После выполнения
```

```
ARP     0                       ARP     4
AR0     300h                    AR0     299h
Память данных                   Память данных
300h    0h                      300h    0h
ACC     0FFFFFFFFh;  C=1        ACC     0h;     C=1
                     OV=X                       OV=0
```

ADDS – Добавление к младшему слову аккумулятора с подавлением расширения знака

СИНТАКСИС Прямая: ADDS dma
Косвенная: ADDS {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) + dma -> ACC
(dma) не знаковое 16-битное число
Зависит от OVM; влияет на C и OV. Не зависит от SXM.

ОПИСАНИЕ Значение адресуемой ячейки памяти данных прибавляется к аккумулятору с подавлением знакового расширения. Данные обрабатываются как 16-битные числа без знака, не зависимо от SXM. Содержание аккумулятора обрабатывается как число со знаком. Бит переноса устанавливается в 1, если результат генерирует перенос; иначе бит C устанавливается 0.
Заметьте, что команда ADDS имеет такой же результат, что и команда ADD при SXM = 0 и сдвиге = 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
ADDS DAT0  ; (DP = 6)
До выполнения                 После выполнения
Память данных                 Память данных
300h    F006h                 300h    F006h
ACC     3h;    C=X            ACC     F009h;  C=0
```

ПРИМЕР 2

```
ADDS *
До выполнения                  После выполнения
ARP     0                      ARP     0
AR0     300h                   AR0     0300h
Память данных                  Память данных
300h    FFFFh                  300h    FFFFh
ACC     7FFF0000h;  C=X        ACC     7FFFFFFFh;   C=0
```

ADDT – Добавление в аккумулятор со сдвигом на значение TREG1

СИНТАКСИС Прямая: ADDT dma
Косвенная: ADDT {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) + [(dma)*$2^{TREG1(3-0)}$] -> (ACC)
Если SXM = 1:
то (dma) знаковое значение.
Если SXM = 0:
то (dma) положительное значение.

Зависит от OVM и SXM; влияет на C и OV.

ОПИСАНИЕ Значение памяти данных сдвигается влево и прибавляется к аккумулятору, результат помещается в аккумулятор. Сдвиг влево определяется четырьмя битами LBS TREG1, - от 0 до 15 бит. Знаковое расширение значения памяти данных контролируется SXM. Бит переноса устанавливается в 1, если результат генерирует перенос; иначе бит C устанавливается 0.
Совместимость программного обеспечения с 1867ВМ2 может быть поддержана путем обнуления бита TRM статусного регистра PMST. Это позволяет любой командой 1867ВМ2, записывающей в TREG0, записывать во все TREG-регистры.Последующие вызовы команды ADDT будут сдвигать значение благодаря содержимому TREG1 (которое такое же, как TREG0),поддерживая объектно-кодовую совместимость.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
ADDT DAT127  ; (DP = 4, SXM = 0)
До выполнения                    После выполнения
Память данных                    Память данных
027h    09h                      027h    09h
TREG1   0FF94h                   TREG1   0FF94h
ACC     0F715h; C=X              ACC     0F7A5h; C=0
```

ПРИМЕР 2

```
ADDT *-,AR4 ; (SXM = 0)
До выполнения                    После выполнения
ARP     0                        ARP     0
AR0     027Fh                    AR0     027Eh
Память данных                    Память данных
027Fh   09h                      027F    09h
TREG1   0FF94h                   TREG1   0FF94h
ACC     0F715h;  C=X             ACC     0F7A5h;   C=0
```

ADRK – Добавление к вспомогательному регистру короткого непосредственного операнда

СИНТАКСИС ADRK #k

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	0	0	8-Bit Constant							

ОПЕРАНДЫ $0 \leq k \leq 255$

ИСПОЛНЕНИЕ (PC) + 1 -> PC
Текущий AR + 8-битовая положительная константа -> текущий AR

ОПИСАНИЕ 8-битовая непосредственная константа прибавляется к текущему дополнительному регистру (на который указывает ARP), результат сохраняется в текущий дополнительный регистр. В ARAU складываются 8-битовые положительные целые значения. Заметьте, что все арифметические операции не знаковые.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР 1 ADRK #80h

До выполнения		После выполнения	
ARP	5	ARP	5
AR5	4321h	AR5	43A1h

AND – Конъюнкция с аккумулятором

СИНТАКСИС

Прямая: AND dma
Косвенная: AND {индекс}[,новый ARP]
Длинная непосредственная: AND #lk[,сдвиг]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	0	1	See Section 5.2						

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	1	1	SHFT†			
16-Bit Constant															

† If the operand and the code are in the same SARAM block

Long immediate addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	1
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
lk: 16-битовая константа
0 ≤ сдвиг ≤ 16

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:
(PC) + 1 -> PC
(ACC(15-0)) AND (dma) -> ACC(15-0)
0 -> ACC(31-16)

Непосредственная адресация:
(PC) + 2 -> PC
ACC(30-0) AND lk*2eсдвиг -> ACC
Не зависит от SXM.

ОПИСАНИЕ

Если используется прямая или косвенная адресация над Мадшим словом аккумулятора и значением памяти данных производится логическую операцию AND, результат сохраняется в позиции младшего слова аккумулятора. Старшее слово аккумулятора обнуляется. Если указывается длинный непосредственный операнд, операнд сдвигается влево и дополнятеся 0 до 31 бита в ладших и старших разрядах Над результатирующим значением и содержанием аккумулятора производят логическую операцию AND.

СЛОВА

1 (Прямая или непосредственная адресация)
2 (Длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Single Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
AND DAT16 ;(DP = 4)
До выполнения                  После выполнения
Память данных                  Память данных
0210h   00FFh                  0210h    00FFh
ACC     12345678h              ACC      00000078h
```

ПРИМЕР 2

```
ADD *
До выполнения                  После выполнения
ARP     0                      ARP     0
AR0     0301h                  AR0     0301h
Память данных                  Память данных
301h    0FF0h                  0301h    0FF00h
ACC     12345678h              ACC     00005600h
```

ПРИМЕР 3

```
AND #00FFh,4
До выполнения                  После выполнения
ACC     12345678h              ACC      00000670h
```

ANDB – Конъюнкция буфера аккумулятора с аккумулятором

СИНТАКСИС ANDB

ОПЕРАНДЫ Нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	0

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) AND (ACCB) -> ACC

ОПИСАНИЕ Над значение аккумулятора и значением буфера аккумулятора выполняется логическая операция AND. Результат сохраняется в аккумуляторе ; буфер аккумулятора не меняется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР ANDB

До выполнения		После выполнения	
ACC	0F0FFFFFh	ACC	05055555h
ACCB	55555555h	ACCB	55555555h

APAC – Прибавляет содержимое P регистра к аккумулятору

СИНТАКСИС APAC

ОПЕРАНДЫ Нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) + (P-регистр со сдвигом) -> ACC
Зависит от OVM и PM; влияет на C и OV.
Не зависит от SXM.

ОПИСАНИЕ Содержание P регистра сдвигается как определено статусными битами PM и прибавляется к содержанию аккумулятора. Результат сохраняется в аккумуляторе. APAC не зависит от бита SXM; P регистр всегда знаковый. Бит переноса устанавливается в 1, если результат генерирует перенос; иначе бит C устанавливается 0.

Команда APAC является подфункцией LTA, LTD, MAC, MACD, MADS, MADD, MPYA и SQRA инструкций.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР
```
APAC  ; (PM = 01)
До выполнения                 После выполнения
P       40h                   P       40h
ACC     20h;     C=X          ACC     A0h;     C=0
```

APL – Конюкция значения памяти данных и значения DBMR или длинной константы

СИТАКСИС прямой APL [#lk,] dma
косвенный APL [#lk,] {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	0	0	dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
lk: 16-битная константа
0 ≤ ,новый ARP ≤ 7

ИСПОЛНЕНИЕ
lk не задан:
(PC) + 1 -> PC
(dma) AND (DMBR) -> (dma)

lk задан:
(PC) + 2 -> PC
(dma) AND lk -> dma

Влияет на TC

ОПИСАНИЕ Если определена длинная непосредственная константа, выполняется AND константы с dma. В противном случае выполняется AND с содержимым DBMR. В любом случае результат записывается в dma. Аккумулятор не меняется. Если результат = 0 TC=1; иначе TC=0.

СЛОВА 1 (второй операнд DBMR)
2 (второй операнд длинная непосредственная константа)

ЦИКЛЫ

Cycles for a Single Instruction (second operand DBMR)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (second operand DBMR)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n-2	2n-2	2n-2, 2n+1†	2n-2+p
External	4n-2+2nd	4n-2+2nd	4n-2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

Cycles for a Repeat (RPT) Execution (long immediate specified)

	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n-1	2n-1	2n-1, 2n+2†	2n-1+2p
External	4n-1+2nd	4n-1+2nd	4n-1+2nd	4n+2+2nd+2p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
APL     #0023h, DAT96   ;DP = 0
До выполнения                   После выполнения
память данных                   память данных
60h     00h;   TC=X             60h     00h;   TC=1
```

ПРИМЕР 2
```
APL     DAT96   ;DP = 0
До выполнения                   После выполнения
DMBR    0ff00h                  DMBR    0ff00h
память данных                   память данных
60h     1111h;   TC=X           60h     1100h;   TC=0
```

ПРИМЕР 3
```
APL     #0100h, *, AR6
До выполнения                   После выполнения
ARP     5                       ARP     6
AR5     300h                    AR5     300h
память данных                   память данных
300h     0fffh;  TC=X           300h     100h; TC=0
```

ПРИМЕР 4
```
APL     *, AR7
До выполнения                   После выполнения
ARP     6                       ARP     7
AR6     310h                    AR6     310h
DMBR    0303h                   DMBR    0303h
память данных                   память данных
310h     0effh;   TC=X          310h     0203h;  TC=0
```

B(D) - Безусловный (задержанный) переход

СИТАКСИС B[D] pma, [,{ind}[, новый ARP]]

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	D	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ 0 ≤ pma ≤ 65536
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ pma -> PC
Модифицирует текщий AR и ARP как задано.

ОПИСАНИЕ Текущий дополнительный регистр и ARP модифицируются заданным образом и управление передается по заданному адресу pma – число или метка. При выполнении задержанного перехода BD перед модификацией PC выполняются две однословных или одна двусловная инструкции следующие за BD.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction			
ROM	DARAM	SARAM	External Memory
4	4	4	4+4p†

Cycles for a Single Instruction для BD			
ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
B PRG191, *+, AR1
```
В PC загружается 191 и выполнение программы продолжается с загруженного адреса. Текущий дополнительный регистр инкрементируется на 1 и ARP устанавливается в 1.

ПРИМЕР 1
```
BD PRG191
MAR     *+, AR1
LDP      #5
```
После модификации текущего AR, APR и DP выполнение программы продолжается с адреса 191.

BACC(D) - Переход (задержанный) по адресу задаваемым аккумулятором

СИНТАКСИС [метка] BACC[D]

ОПЕРАНДЫ нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	D

ВЫПОЛНЕНИЕ ACC(15-0) -> PC

ОПИСАНИЕ Управление передается по 16-битовому адресу, взятому из младшего слова аккумулятора. При выполнении задержанного перехода BACCD перед модификацией PC выполняются две однословных или одна двусловная инструкции следующие за BACCD.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

Cycles for a Single Instruction для BACCD

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

ПРИМЕР 1

```
BACC    ; (ACC содержит 191)
```

В PC загружается 191 и выполнение программы загруженного адреса.

ПРИМЕР 1

```
BACCD    ; (ACC содержит 191)
MAR     *+, AR1
LDP      #5
```

После модификации текущего AR, APR и DP выполнение программы продолжается с адреса 191

BANZ(D) – Переход(задержанный), если дополнительный регистр ≠ 0

СИТАКСИС BANZ[D] pma, [,{ind}[, новый ARP]]

ОПЕРАНДЫ $0 \le pma \le 65536$
$0 \le$ новый ARP ≤ 7

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	D	1	1	1	See Section 5.2						
16-Bit Constant															

ВЫПОЛНЕНИЕ
```
if( текущий AR != 0 )
        pma -> PC;
else
        (PC) + 2 -> PC;
```
Модификацирует текущий AR и ARP как задано

ОПИСАНИЕ Управление передается по адресу pma, если содержимое текущего AR ≠ 0. В противном случае управление передается к следующей инструкции. По умолчанию значени текущего AR уменьшается на 1. Для выполнения N итераций цикла счетчик цикла перед входом в цикл устанавливают в N-1. При выполнении задержанного перехода BANZD перед модификацией PC выполняются две однословных или одна двусловная инструкции следующие за BANZD.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

Cycles for a Single Instruction для BANZD

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
BANZ  PGM0
До выполнения                   После выполнения
ARP     0                       ARP    0
AR0     5h                      AR0    4h
```
В PC загружается 0 и выполнение программы продолжается с адреса 0.
ИЛИ
```
До выполнения                   После выполнения
ARP     0                       ARP    0
AR0     0h                      AR0    0ffffh
```
Значение PC увеличивается на 2 и выполнение программы продолжается с адреса, указанного в PC.

ПРИМЕР 2
```
BANZD PGM0
LACC  #01h
LDP   #5
До выполнения                   После выполнения
ARP     0                       ARP    0
AR0     5h                      AR0    4h
```

```
                DP      4                       DP      5
                ACC     0                       ACC     1h
                После модификации DP и ACC программа PC устанавливается в 0.

ПРИМЕР 3        MAR     *, AR0
                LAR     AR1, #3
                LAR     AR0, #60H
        PGM191  ADD     *+, AR1
                BANZ    PGM191, AR0
                Содержимое памяти данных 60h - 63h прибавляется
                к аккумулятору.
```

BCND(D) – Переход (задержанный) по условию

СИТАКСИС BCND[D] pma, [усл1][,усл2][,…

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	D	0	0	TP†		ZLVC†				ZLVC†			
16-Bit Constant															

ОПЕРАНДЫ 0 ≤ pma ≤ 65536

Условия:

ACC=0	EQ
ACC!=0	NEQ
ACC<0	LT
ACC<=0	LEQ
ACC>0	GT
ACC>=0	GEQ
C=0	NC
C=1	C
OV=0	NOV
OV=1	OV
BIO младшие	BIO
TC=0	NTC
TC=1	TC
Без условия	UNC

ВЫПОЛНЕНИЕ
```
If( условие(я) )
    Then pma -> PC
    Else PC + 2 -> PC
```

ОПИСАНИЕ Управление передается по адресу pma, если условие истинно. Заметьте, что не все комбинации условий значимы. Так же заметьте, что тестирование BIO сочетается исключительно с тестированием TC.
Две однословные команды или две двухсловные команды, следующие за переходом вызываются из программной памяти и выполняются до перехода, если установлен задержанный переход (определяется суффиксом «D»).

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

Cycles for a Single Instruction для BCNDD

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
BCND  PG191, LEQ, C
```
Если содержание аккумулятора меньше или равно нулю и бит переноса установлен в 1, то программный адрес 191 загружается в PC и программа продолжает выполняться с этого адреса. Если условие не выполняется , то выполнение продолжается с адреса PC + 2.

ПРИМЕР 2
```
BCNDD  PG191, OV
```

```
MAR   *,AR1
LDP   #5
```

После изменения AR, ARP и DP исполнение программы продолжается с адреса 191, если флаг переполнения (OV) в статусном регистре ST0 установлен в 1. Если флаг не установлен, то выполнение продолжается с команды следующей за командой LDP.

BIT - Тестирование бита

СИТАКСИС Прямая: BIT dma, битовый код
Косвенная: BIT {индекс}, битовый код [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	BITX†				0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	BITX†				1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ pma ≤ 65536
0 ≤ новый ARP ≤ 7
0 ≤ битовый код ≤ 15

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(dma-бит(15-битовый код)) -> TC

Влияет на TC.

ОПИСАНИЕ Команда BIT копирует значения бита слова памяти данных в TC бит статусного регистра ST1.Заметьте, что BIT, CMPR, LST1, APL, CPL, OPL, XPL и NORM команды тоже влияют на TC бит статусного регистра ST1. Значение битового кода которое соответствует битовому адресу, представлено в следующей таблице:

Битовый адрес	Битовый код
(LSB) 0	1111
1	1110
2	1101
3	1100
4	1001
5	1010
6	1001
7	1000
8	0111
9	0110
10	0101
11	0100
12	0011
13	0010
14	0001
(MSB) 15	0000

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
BIT  0h,15  ; (DP = 6). Тестируется младший бит ячейки 300h
До выполнения                    После выполнения
Память данных                    Память данных
300h    4DC8h                    300h   04DC8h
TC      0                        TC     0
```

ПРИМЕР 2

```
BIT  *,0,AR1 . Тестируется старший бит ячейки 310h
До выполнения                    После выполнения
ARP     0                        ARP     1
AR0     310h                     AR0     310h
Память данных                    Память данных
310h    8000h                    310h   8000h
TC      0                        TC     1
```

BITT - Тестирование бита по значению TREG2

СИТАКСИС Прямая: BITT dma
Косвенная: BITT {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(dma-бит(15-TREG2(3-0))) -> TC

Влияет на TC.

ОПИСАНИЕ Команда BITT копирует значение бита слова памяти данных в TC бит статусного регистра ST1.Заметьте, что BIT, CMPR, LST1, APL, CPL, OPL, XPL и NORM команды тоже влияют на TC бит статусного регистра ST1. Битовый адрес, определенный значением битового кода, содержится в 4-х младших битах TREG2, представлен ниже в таблице.

Битовый адрес	Битовый код
(LSB) 0	1111
1	1110
2	1101
3	1100
4	1001
5	1010
6	1001
7	1000
8	0111
9	0110
10	0101
11	0100
12	0011
13	0010
14	0001
(MSB) 15	0000

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
BITT  00h  ; (DP = 6). Тестируется 14 бит ячейки данных 300h
                До выполнения                  После выполнения
                Память данных                  Память данных
                300h    4DC8h                  300h   04DC8h
                TREG2   1h                     TREG2   1h
                TC      0                      TC      1
```

ПРИМЕР 2

```
BITT  *  ; (DP = 6). Тестируется 1 бит ячейки данных 310h
                До выполнения                  После выполнения
                ARP     1                      ARP     1
                AR1     310h                   AR1     310h
                Память данных                  Память данных
                310h    8000h                  310h   8000h
                TREG2   0Eh                    TREG2   0Eh
                TC      0                      TC     0
```

BLDD – Перемещение блока данных из памяти данных в память данных

СИТАКСИС

Ниже список всех вариантов следующего основного синтаксиса: BLDD scr, dst

Прямой K/DMA	BLDD #lk,dma
Косвенный K/DMA	BLDD #lk, {ind} [,новый ARP]
Прямой DMA/K	BLDD dma,#lk
Косвенный DMA/K	BLDD {ind},#lk [,новый ARP]
Прямой BMAR/DMA	BLDD BMAR,dma
Косвенный BMAR/DMA	BLDD BMAR,{ind}[,новый ARP]
Прямой DMA/BMAR	BLDD dma,BMAR
Косвенный DMA/BMAR	BLDD {ind},BMAR[,новый ARP]

КОД КОМАНДЫ

Direct addressing with SRC by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	0	0	dma						

Indirect addressing with SRC by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	0	1	See Section 5.2						

Direct addressing with DEST specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	dma						

Indirect addressing with DEST specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	1	See Section 5.2						

Direct addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	0	0	dma						
16-Bit Constant															

Indirect addressing with DEST specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

$0 \le lk \le 65535$
$0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ИСПОЛНЕНИЕ

(PFC) -> MCS
Если длинная непосредственная
    (PC) + 2 -> PC
Иначе:
    (PC) + 1 -> PC

(BMAR) -> PFC
Пока (счетчик повторений) != 0:
    (scr адресуемый PFC) - > (dst/scr адресуемый PFC)
    Изменение текущего AR и ARP как определено,
    (PFC) + 1 -> PFC

(счетчик повторений) - 1 -> счетчик повторений
иначе
(scr адресуемый PFC) - > (dst/scr адресуемый PFC)
Изменение AR и ARP как определено,
(MSC) -> PFC

ОПИСАНИЕ

Слово из памяти данных на которое указывает scr копируется в память данных, на которое указывает dst. Слово источника и/или приемника может быть указано длинным непосредственным операдом, значением регистра BMAR, или адресом памяти данных. Заметьте, что не все scr/dst комбинации типов ссылок имеют смысл. Цикл RPT может быть использован с командой BLDD в режиме косвенной адресации для перемещения блоков данных. Число слов, которые будут перемещены, на единицу больше, чем число, содержащееся в счетчике повторений RPTC перед выполнением команды. Если при выполнении инструкции в цикле RPT данные адресуются длинной непосредственной константой или через BMAR, то адрес данных автоматически инкрементируется. При выполнении инструкции в цикле RPT dma адрес не инкрементируется автоматичестки.
Заметьте, что блоки источника и приемника не должны быть полностью во внутренней или внешней памяти. Прерывания запрещены в течение исполнения операции BLDD в цикле RPT.

Длинный непосредственный операнд и/или значение BMAR не применимы для адресации внутрикристальных регистров картированных в памяти. Для адресации внутрикристальных регистров картированных в памяти используются прямая и косвенная адресации.

СЛОВА

1 (источник или приемник задается через регистр BMAR)
2 (источник или приемник задается длинным непос-редственным операндом)

ЦИКЛЫ

Cycles for a Single Instruction (SRC or DEST in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	2	2	2	2+p
Source: SARAM Destination: DARAM	2	2	2	2+p
Source: External Destination: DARAM	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$2+d_{src}+p$
Source: DARAM Destination: SARAM	2	2	2, 3†	2+p
Source: SARAM Destination: SARAM	2	2	2, 3†	2+p
Source: External Destination: SARAM	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$ $3+d_{src}$†	$2+d_{src}+p$

Source: DARAM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p$
Source: SARAM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p$
Source: External Destination: External	$3+d_{src}+d_{dst}$	$3+d_{src}+d_{dst}$	$3+d_{src}+d_{dst}$	$5+d_{src}+d_{dst}+p$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC or DEST in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	n+1	n+1	n+1	n+1+p
Source: SARAM Destination: DARAM	n+1	n+1	n+1	n+1+p
Source: External Destination: DARAM	$n+1+nd_{src}$	$n+1+nd_{src}$	$n+1+nd_{src}$	$n+1+nd_{src}+p$
Source: DARAM Destination: SARAM	n+1	n+1	n+1, n+3†	n+1+p
Source: SARAM Destination: SARAM	n+1, 2n-1‡	n+1, 2n-1‡	n+1, 2n-1‡, n+3§, 2n+1§	n+1+p, 2n-1+p‡
Source: External Destination: SARAM	$n+1+nd_{src}$†	$n+1+nd_{src}$	$n+1+nd_{src}$, $n+3+nd_{src}$†	$n+1+nd_{src}+p$
Source: DARAM Destination: External	$2n+1+nd_{src}$	$2n+1+nd_{src}$	$2n+1+nd_{src}$	$2n+1+nd_{src}+p$
Source: SARAM Destination: External	$2n+1+nd_{src}$	$2n+1+nd_{src}$	$2n+1+nd_{src}$	$2n+1+nd_{src}+p$
Source: External Destination: External	$4n-1+nd_{src}+nd_{dst}$	$4n-1+nd_{src}+nd_{dst}$	$4n-1+nd_{src}+nd_{dst}$	$4n+1+nd_{src}+nd_{dst}+p$

Cycles for a Single Instruction (SRC or DEST long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	3	3	3	3+2p
Source: SARAM Destination: DARAM	3	3	3	3+2p
Source: External Destination:	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$	$3+d_{src}+2p$

DARAM				
Source: DARAM Destination: SARAM	3	3	3, 4†	3+2p
Source: SARAM Destination: SARAM	3	3	3, 4†	3+2p
Source: External Destination: SARAM	3+d_{src}	3+d_{src}	3+d_{src}, 4+d_{src}	3+d_{src}+2p
Source: DARAM Destination: External	4+d_{dst}	4+d_{dst}	4+d_{dst}	6+d_{dst}+2p
Source: SARAM Destination: External	4+d_{dst}	4+d_{dst}	4+d_{dst}	6+d_{dst}+2p
Source: External Destination: External	n+2	n+2	n+2	n+2+2p

Cycles for a Repeat (RPT) Execution (SRC or DEST long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: DARAM	n+2	n+2	n+2	n+2+2p
Source: External Destination: DARAM	n+2+nd_{src}	n+2+nd_{src}	n+2+nd_{src}	n+2+nd_{src}
Source: DARAM Destination: SARAM	n+2	n+2	n+2, n+4†	n+2+2p
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, n+4§, 2n+2§	n+2+2p, 2n+2p‡
Source: External Destination: SARAM	n+2nd_{src}	n+2nd_{src}	n+2nd_{src}, n+4+nd_{src}†	n+2+nd_{src}+2p
Source: DARAM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}+2p
Source: SARAM Destination: External	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}	2n+2+nd_{dst}+2p
Source: External Destination: External	4n+nd_{src}+nd_{dst}‡	4n+nd_{src}+nd_{dst}	4n+nd_{src}+nd_{dst}	4n+2+nd_{src}+nd_{dst}+2p

ПРИМЕР 1

```
BLDD #300h,20h  ; (DP = 6)
```

До выполнения память данных	После выполнения память данных

```
                  300h     0h                    300h     0h
                  320h     0Fh                   320h     0h
```

ПРИМЕР 2

```
                  BLDD  *+,#321h,AR3
                  До выполнения                  После выполнения
                  ARP      2                     ARP      3
                  AR2      301h                  AR2      302h
                  память данных                  память данных
                  301h     01h                   301h     01h
                  321h     0Fh                   321h     01h
```

ПРИМЕР 3

```
                  BLDD  BMAR,*
                  До выполнения                  После выполнения
                  ARP      2                     ARP      2
                  BMAR     320h                  BMAR     320h
                  AR2      340h                  AR2      340h
                  память данных                  память данных
                  320h     01h                   320h     01h
                  340h     0Fh                   340h     01h
```

ПРИМЕР 4

```
                  BLDD  00h,BMAR  ; (DP = 6)
                  До выполнения                  После выполнения
                  300h     0Fh                   300h     0Fh
                  BMAR     320h                  BMAR     320h
                  память данных                  память данных
                  320h     01h                   320h     0Fh
```

ПРИМЕР 5

```
                  RPT  2
                  BLDD  #300h,*+
                  До выполнения                  После выполнения
                  ARP      0                     ARP      0
                  AR0      320h                  AR0      323h
                  память данных                  память данных
                  300h     7F98h                 300h     7F98h
                  301h     0FFE6h                301h     0FFE6h
                  302h     9522h                 302h     9522h
                  320h     8DEEh                 320h     7F98h
                  321h     9315h                 321h     0FFE6h
                  322h     2531h                 322h     9522h
```

BLDP – Перемещение блока данных из памяти данных в память программ.

СИТАКСИС Прямой : BLDP dma
Косвенный : [BLDP {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ
(PC) + 1 -> PC
(PFC) -> MCS
(BMAR) -> PFC

Пока (счетчик повторений) != 0:
dma -> (dst адресуемый PFC)
Изменение AR и ARP как определено,
(PFC) + 1 -> PFC
(счетчик повторений) - 1 -> счетчик повторений.
иначе
dma -> (dst адресуемый PFC)
Изменение AR и ARP как определено.
(MSC) -> PFC

ОПИСАНИЕ Слово из памяти данных копируется по адресу памяти программ,указанному в BMAR. Цикл RPT может быть использован с командой BLDP в режиме косвенной адресации для перемещения блоков данных.
Число слов, которые будут перемещены, на единицу больше, чем число, содержащееся в счетчике повторений RPTC перед выполнением команды. Значение регистра BMAR автоматически инкрементируется.
. Заметьте, что блоки источника и приемника не должны быть полностью во внутренней или внешней памяти. Прерывания запрещены в течение исполнения операции BLDD с использованием инструкции RPT .

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination : DARAM	2	2	2	2+p
Source: SARAM Destination : DARAM	2	2, 3¶	2	2+p
Source: External Destination : DARAM	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

Source: DARAM Destination : SARAM	2	2	2, 3†	2+p
Source: SARAM Destination : SARAM	2	2	2, 3†¶, 4§	2+p
Source: External Destination : SARAM	2+d_{src}	2+d_{src}	2+d_{src},3+d_{src} †	3+d_{src}+p_{code}
Source: DARAM Destination : External	3+p_{dst}	3+p_{dst}	3+p_{dst}	4+p_{dst}+p_{code}
Source: SARAM Destination : External	3+p_{dst}	3+p_{dst}	3+p_{dst}, 3+p_{dst}¶	4+p_{dst}+p_{code}
Source: External Destination : External	3+d_{src}+p_{dst}	3+d_{src}+p_{dst}	3+d_{src}+p_{dst}	5+d_{src}+p_{dst}+p_{code}

† If the destination operand and the code are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination : DARAM	n+1	n+1	n+1	n+1+p_{code}
Source: SARAM Destination : DARAM	n+1	n+1	n+1, n+2¶	n+1+p_{code}
Source: External Destination : DARAM	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}	n+2+nd_{src}+p_{code}
Source: DARAM Destination : SARAM	n+1	n+1	n+1, n+2†	n+1+p_{code}
Source: SARAM Destination : SARAM	n+1,2n-1‡	n+1,2n-1‡	n+1,2n-1‡, n+2†¶,2n+1§	n+1+p_{code}, 2n-1+p_{code}‡
Source: External Destination : SARAM	n+1+nd_{src}	n+1+nd_{src}	n+1+nd_{src}, n+2+np_{src}†	n+2+nd_{src}+p_{code}
Source: DARAM Destination : External	2n+1+np_{dst}	2n+1+np_{dst}	2n+1+np_{dst}	2n+2+np_{dst}+p_{code}
Source: SARAM Destination : External	2n+1+np_{dst}	2n+1+np_{dst}	2n+1+np_{dst}, 2n+2+np_{dst}¶	2n+2+np_{dst}+p_{code}

Source: External Destination : External	4n-1+nd_{src}+np_{dst}	4n-1+nd_{src}+np_{dst}	4n-1+nd_{src}+np_{dst}	4n+1+nd_{src}+np_{dst}+p_{code}

ПРИМЕР 1

```
BLDP 00h      ; (DP = 6)
```

До выполнения		После выполнения	
память данных		память данных	
300h	0A089h	300h	0A089h
BMAR	2800h	BMAR	2800h
Программная память		Программная память	
2800h	1234h	2800h	0A089h

ПРИМЕР 2

```
BLDP *,AR0
```

До выполнения		После выполнения	
ARP	7	ARP	0
AR7	310h	AR7	310h
память данных		память данных	
310h	0F0F0h	310h	0F0F0h
BMAR	2800h	BMAR	2800h
Программная память		Программная память	
2800h	1234h	2800h	0F0F0h

BLPD – Перемещение блока данных из памяти программ в память данных

СИТАКСИС

Ниже список всех вариантов следующего основного синтаксиса: BLPD scr, dst

Прямой K/DMA	BLPD #lk,dma
Косвенный K/DMA	BLPD #lk, {ind} [,новый ARP]
Прямой BMAR/DMA	BLPD BMAR,dma
Косвенный BMAR/DMA	BLPD BMAR,{ind}[,новый ARP]

Direct addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	0	0	dma						

Indirect addressing with SRC specified by BMAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	0	1	See Section 5.2						

Direct addressing with SRC specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	0	Dma						
16-Bit Constant															

Indirect addressing with DEST specified by long immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ lk ≤ 65535
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ

```
Если длинная непосредственная:
        (PC) + 2 -> PC
        (PFC) -> MCS
        lk -> PFC
Иначе:
        (PC) + 1 -> PC
        (PFC) -> MSC
        (BMAR) -> PFC

Пока (счетчик повторений) != 0:
        (pma адресуемый PFC) - > dst
        Изменение AR и ARP как определено,
        (PFC) + 1 -> PFC
        (счетчик повторений) – 1 -> счетчик повторений.
иначе
(pma адресуемый PFC) - > dst
Изменение AR и ARP как определено,
(MSC) -> PFC
```

ОПИСАНИЕ

Слово из памяти программ, на которое указывает scr, копируется по адресу памяти данных, на которое указывает dst. Источник может быть задан длинным непосредственной операндом или через BMAR. Приемник адресуется либо через dma, либо через значение текущего дополнительного регистра. Цикл RPT может быть использован с командой BLPD в режиме косвенной адресации для перемещения блоков данных.
Число слов, которые будут перемещены, на единицу

больше, чем число, содержащееся в счетчике повторений RPTC перед выполнением команды. Если при выполнении инструкции в цикле RPT источник адресуется длинной непосредственной константой или через BMAR, то адрес источника автоматически инкрементируется.
Заметьте, что не все src/dst комбинации указателей разрешены. Заметьте, что блоки источника и приемника не должны быть полностью во внутренней или внешней памяти. Прерывания запрещены в течение повторения команды BLPD.

СЛОВА 1 (Источник определен регистром BMAR)
2 (Источник определен длинным непосредственным операндом)

ЦИКЛЫ

Cycles for a Single Instruction (SRC in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	2	2	2	$2+p_{code}$
Source: SARAM Destination: DARAM	2	2	2	$2+p_{code}$
Source: External Destination: DARAM	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$	$2+p_{src}+p_{code}$
Source: DARAM/ROM Destination: SARAM	2	2	2, 3†	$2+p_{code}$
Source: SARAM Destination: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External Destination: SARAM	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$, $3+p_{src}$†	$2+p_{src}+p_{code}$
Source: DARAM/ROM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p_{code}$
Source: SARAM Destination: External	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+p_{code}$
Source: External Destination: External	$3+p_{src}+d_{dst}$	$3+p_{src}+d_{dst}$	$3+p_{src}+d_{dst}$	$5+p_{src}+d_{dst}+p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC in BMAR)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+1	n+1	n+1	$n+1+p_{code}$

Source: SARAM Destination: DARAM	n+1	n+1	n+1	n+1+p_{code}
Source: External Destination: DARAM	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}+p_{code}
Source: DARAM/ROM Destination: SARAM	n+1	n+1	n+1, n+3t	n+1+p_{code}
Source: SARAM Destination: SARAM	n+1,2n-1†	n+1, 2n-1†	n+1,2n-1†, n+3†, 2n+1§	n+1+p_{code}, 2n-1+p_{code}‡
Source: External Destination: SARAM	n+1+np_{src}	n+1+np_{src}	n+1+np_{src}, n+3+np_{src}†	n+1+np_{src}+p_{code}
Source: DARAM/ROM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p_{code}
Source: SARAM Destination: External	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}	2n+1+nd_{dst}+p_{code}
Source: External Destination: External	4n-1+np_{src}+nd_{dst}	4n-1+np_{src}+nd_{dst}	4n-1+np_{src}+nd_{dst}	4n+1+np_{src}+nd_{dst} +p_{code}

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

Cycles for a Single Instruction (SRC long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	3	3	3	3+2p_{code}
Source: SARAM Destination: DARAM	3	3	3	3+2p_{code}
Source: External Destination: DARAM	3+p_{src}	3+p_{src}	3+p_{src}	3+p_{src}+2p_{code}
Source: DARAM/ROM Destination: SARAM	3	3	3, 4†	3+2p_{code}
Source: SARAM Destination: SARAM	3	3	3,4†	3+2p_{code}
Source: External Destination: SARAM	3+p_{src}	3+p_{src}	3+p_{src}, 4+p_{src}†	3+p_{src}+2p_{code}

Source: DARAM/ROM Destination: External	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+2p_{code}$
Source: SARAM Destination: External	$4^{+}d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+2p_{code}$
Source: External Destination: External	$4+p_{src}{}^{+}d_{dst}$	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$6+p_{src}+d_{dst}+2p_{code}$

† If the destination operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (SRC long immediate)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
Source: External Destination: DARAM	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}+2p_{code}$
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2, n+4†	$n+2+2p_{code}$
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, n+4†, 2n+2§	$n+2+2p_{code}$, $2n+2P_{code}$‡
Source: External Destination: SARAM	$n+2+np_{src}$†	$n+2+np_{src}$	$n+2+np_{src}$, $n+4+np_{src}$†	$n+2+np_{src}+2p_{code}$
Source: DARAM/ROM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}+2p_{code}$
Source: SARAM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}+2p_{code}$
Source: External Destination: External	$4n+np_{src}+nd_{dst}$†	$4n+np_{src}+nd_{dst}$	$4n+np_{src}+nd_{dst}$	$4n+2+np_{src}+nd_{dst}$

† If the destination operand and the code are in the same SARAM block
‡ If both the source and the destination operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block

ПРИМЕР 1

```
BLPD #800h,00h  ; (DP = 6)
```

До выполнения		После выполнения	
Программная память		Программная память	
800h	0Fh	800h	0Fh
память данных		память данных	
300h	0h	300h	0h

ПРИМЕР 2

```
BLPD  #800h,*,AR7
```

До выполнения		После выполнения	
ARP	0	ARP	7
AR0	310h	AR0	310h
Программная память		Программная память	
800h	1111h	800h	1111h
память данных		память данных	
310h	0100h	310h	1111h

ПРИМЕР 3

```
BLPD  BMAR,00h  ; (DP = 6)
```

До выполнения		После выполнения	
BMAR	800h	BMAR	800h
Программная память		Программная память	
800h	0Fh	800h	0Fh
память данных		память данных	
300h	0h	300h	0Fh

ПРИМЕР 4

```
BLPD  BMAR,*+,AR7
```

До выполнения		После выполнения	
ARP	0	ARP	7
AR0	300h	AR0	301h
BMAR	810h	BMAR	810h
Программная память		Программная память	
810h	4444h	810h	4444h
память данных		память данных	
300h	0100h	300h	4444h

BSAR – Циклический сдвиг

СИТАКСИС BSAR сдвиг

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	SHFT†			

ОПЕРАНДЫ 0 ≤ сдвиг ≤ 16

ИСПОЛНЕНИЕ (PC) + 1 -> PC
$(ACC)/2^{сдвиг}$ -> ACC

Влияет на SXM.

ОПИСАНИЕ Команда BSAR исполняет от 1 до 16 бит циклический арифметический сдвиг вправо аккумулятора за один цикл. Знак результата сдвига определяется битом SXM. Если SXM = 0 старшие биты аккумулятора заполняются 0; иначе старшие биты аккумулятор являются знаковы расширением значения аккумулятора.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
BSAR 16        ; (SXM = 0)
До выполнения                  После выполнения
ACC     00010000h              ACC     00000001h
```

ПРИМЕР 2
```
BSAR 4         ; (SXM = 1)
До выполнения                  После выполнения
ACC     0FFF10000h             ACC     0FFFF1000h
```

CALA(D) – Вызов (задержанный) подпрограммы из по адресу задаваемым аккумулятором

СИТАКСИС [метка] CALA[D]

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0

CALAD

1	0	1	1	1	1	1	0	0	0	1	1	1	1	0	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ Без задержки: PC + 1 -> TOS
С задержкой: PC + 3 -> TOS

ACC(15-0) -> PC

ОПИСАНИЕ

Текущее значение программного счетчика (PC) инкрементируется и заносится на верхушку стека (TOS). Потом содержание младшей половины аккумулятора загружается в PC. Исполнение программного продолжается с этого адреса. Если используется задержанный вызов (устанавливается суффиксом «D»), то одна двухсловная команда или две однословные команды, следующие за командой вызова,выполняются до выполнения вызова.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

Cycles for a Single Instruction для CALAD

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

ПРИМЕР 1

```
CALA
До выполнения                 После выполнения
PC      25h                   PC      83h
ACC     83h                   ACC     83h
TOS     100h                  TOS     26h
```

ПРИМЕР 2

```
CALAD
MAR *+,AR1
LDP     #5
До выполнения                 После выполнения
ARP     0                     ARP     1
AR0     8                     AR0     9
DP      0                     DP      5
PC      25h                   PC      83h
ACC     83h                   ACC     83h
TOS     100h                  TOS     28h
```

После изменения AR,ARP и DP адрес команды следующей за командой LDP заносится в стек и исполнение программы продолжается с адреса 83h.

CALL(D) – Безусловный вызов (задержанный)

СИТАКСИС CALL[D] pma,{ind}[,новый ARP]

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0	1	See Section 5.2						
16-Bit Constant															

CALLD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ 0 ≤ pma ≤ 65535
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ Без задержки: PC + 2 -> TOS
С задержкой: PC + 4 -> TOS
pma -> PC
Изменение текущего AR и ARP как определено.

ОПИСАНИЕ Текущее значение программного счетчика (PC) инкрементируется и заносится на верхушку стека (TOS). Потом значение, адресуемое программной памятью (pma) загружается в PC.
Текущий дополнительный регистр и ARP изменяются как определено. Если используется вызов с задержкой (устанавливается суффиксом «D»), то одна двухсловная команда или две однословные команды, следующие за командой вызова, выполняются до выполнения вызова.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+4p

Cycles for a Single Instruction для CALLD

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
CALL  PRGG191,*+,AR0
До выполнения                  После выполнения
ARP     1                      ARP     0
AR1     05h                    AR1     06h
PC      30h                    PC      0BFh
TOS     100h                   TOS     32h
```

0BFh загружается в программный счетчик и исполнение программы продолжается с этого адреса.

ПРИМЕР 2

```
CALLD  PRG191
MAR    *+,AR1
LDP    #5
До выполнения                  После выполнения
ARP     0                      ARP     1
AR0     09h                    AR0     0Ah
DP      1                      DP      5
PC      30h                    PC      0BFh
TOS     100h                   TOS     34h
```

После изменения AR,ARP и DP адрес команды следующей

за командой LDP заносится в стек и исполнение программы продолжается с адреса 0BFh.

CC(D) - Вызов (задержанный) по условию

СИТАКСИС CC[D] pma[,условие1][,условие2][…

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	TP†		ZLVC†				ZLVC†			
16-Bit Constant															

CCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	TP†		ZLVC†				ZLVC†			
16-Bit Constant															

ОПЕРАНДЫ 0 ≤ pma ≤ 65535

Условия:

ACC = 0	EQ
ACC!=0	NEQ
ACC<0	LT
ACC<=0	LEQ
ACC>0	GT
ACC>=0	GEQ
C=0	NC
C=1	C
OV=0	NOV
OV=1	OV
BIO младшие	BIO
Без условия	UNC

ИСПОЛНЕНИЕ

```
Если (условия(е))
        Без задержки:   PC + 2 -> TOS
        С задержкой:    PC + 4 -> TOS
        pma -> PC
Иначе
        PC + 2 -> PC
```

ОПИСАНИЕ Если заданное условие выполняется, управление передается по адресу pma. Текущее значение программного счетчика инкрементируется и заносится на верхушку стека (TOS). Заметьте, что не все комбинации условий имеют смысл. К тому же, условия NTC, TC и BIO взаимно исключаются. Если используется вызов с задержкой (устанавливается суффиксом «D»), то одна двухсловная команда или две однословные команды, следующие за командой вызова, выполняются до выполнения вызова. Команда CC аналогична команде CALL, когда все условия истинны.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+4p†
False	2	2	2	2+2p

Cycles for a Single Instruction для CCD

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
CC    PRGG191,LEQ,C
```

Если содержимое аккумулятора меньше или равно нулю и установлен в 1 бит переноса, то 0BFh загружается в программный счетчик и исполнение программы продолжается с этого адреса. Если условия не выполнились, то выполнение продолжается с команды следующей за командой CC.

ПРИМЕР 2

```
CCD     PRG191,LEQ,C
MAR     *+,AR1
LDP     #5
```

Текущие AR,ARP и DP изменяются, как определено. Если содержимое аккумулятора меньше или равно нулю и установлен в 1 бит переноса, то адрес команды следующей за командой LDP загружается в стек и исполнение программы продолжается с адреса 0BFh. Если условия не выполнились, то выполнение продолжается с команды следующей за командой LDP.

CLRC – Очистка контрольного бита

СИТАКСИС CLRC контрольный бит

КОД КОМАНДЫ

CLRC OVM (Clear overflow mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0

CLRC SXM (Clear sign Extension mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	1	0

CLRC HM (Clear hold mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	0	0

CLRC TC (Clear test/control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	1	0

CLRC C (Clear carry)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0

CLRC CNF (Clear configuration control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	0	0

CLRC INTM (Clear interrupt mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0

CLRC XF (Clear external flag pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	0	0

ОПЕРАНДЫ контрольный бит: ST0, ST1 биты (из следующего набора):
{C, CNF, HM, INTM, OVM, TC, SXM, XF}

ИСПОЛНЕНИЕ (PC) + 1 -> PC
0 -> контрольный бит

ОПИСАНИЕ Указанный контрольный бит обнуляется. Заметьте, что команда LST может быть также использована для загрузки ST0 и ST1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 CLRC TC ; TC - 11-й бит ST1

До выполнения		После выполнения	
ST1	x9xxh	ST1	x1xxh

CMPL - Инверсия аккумулятора

СИТАКСИС CMPL

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(~ACC) -> ACC

ОПИСАНИЕ Содержимое аккумулятора переписывается с логической инверсией (дополнение до единицы). Бит переноса не влияет.

СЛОВА 1

ЦИКЛЫ

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 CMPL

До выполнения	После выполнения
ACC 0F7982513h; C=X	ACC 0867DAECh; C=X

CMPR - Сравнение дополнительного регистра с ARCR

СИТАКСИС CMPR CM

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	1	CM†	

ОПЕРАНДЫ $0 \leq CM \leq 3$

ИСПОЛНЕНИЕ (PC) + 1 -> PC
Результат сравнения текущего AR с ARCR заносится в TC бит статусного регистра ST1.

Влияет на TC; зависит от NDX.
Не зависит и не влияет на SXM.

ОПИСАНИЕ Команда CMPR выполняет сравнения заданные значением CM:

При CM = 00 проверка на AR = ARCR
При CM = 01 проверка на AR < ARCR
При CM = 10 проверка на AR > ARCR
При CM = 11 проверка на AR != ARCR

Если условие истинно, то в TC бит загружается 1. Если условие ложно, то в TC бит загружается 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 CMPR 2

До выполнения		После выполнения	
APR	4	APR	4
ARCR	0FFFFh	ARCR	0FFFFh
AR4	7FFFh	AR4	7FFFh
TC	1	TC	0

CPL – Сравнение DBMR или длинного непосредственного значения со значением слова памяти данных

СИТАКСИС

Прямая: CPL [,#lk]dma
Косвенная: CPL [,#lk]{ind}[,новый ARP]

КОД КОМАНДЫ

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	1	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	0	Dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
lk: 16-ти битовая константа
0 ≤ новый ARP ≤ 7

ИСПОЛНЕНИЕ

lk не задана
(PC) + 1 -> PC
Сравнение содержания DMBR с (dma)
Если (DMBR) = (dma),
TC = 1;
Иначе,
TC = 0.

Lk задана
(PC) + 2 -> PC
Сравнение lk с (dma).
Если lk = (dma),
TC = 1;
Иначе,
TC = 0.

Влияет на TC.
Не зависит от SXM.

ОПИСАНИЕ

Если два сравниваемых значения равны, то в TC бит заносится 1. В противном случае TC бит обнуляется.

СЛОВА

1 (Если длинное непосредственное значение не задано)
2 (Если длинное непосредственное значение задано)

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p

SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2, 3†	2+2p
External	2+d	2+d	2+d	3+d+2p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	n+1	n+1	n+1, n+2†	n+1+2p
External	n+1	n+1	n+1	n+2+2p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
CPL  #60h,60h
До выполнения                 После выполнения
Память данных                 Память данных
60h     066h                  60h     066h
TC      1                     TC      0
```

ПРИМЕР 2
```
CPL  60h
До выполнения                 После выполнения
Память данных                 Память данных
60h     066h                  60h     066h
DMBR    066h                  DMBR    066h
TC      0                     TC      1
```

ПРИМЕР 3
```
CPL  #0F1h,*,AR6
До выполнения                 После выполнения
ARP     7                     ARP     6
AR7     300h                  AR7     300h
Память данных                 Память данных
300h    0F1h                  300h    0F1h
TC      1                     TC      1
```

ПРИМЕР 4
```
CPL  *,AR7
До выполнения                 После выполнения
ARP     6                     ARP     7
```

```
AR6      300h
Память данных
300h     0F1h
DMBR     0F0h
TC       0
```

```
AR6      300h
Память данных
300h     0F1h
DMBR     0F0h
TC       0
```

CRGT – Тестирование ACC > ACCB

СИТАКСИС CRGT

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
Если (ACC) > (ACCB)
To (ACC) -> (ACCB); 1 -> C
Если (ACC) < (ACCB)
To (ACCB) -> (ACC); 0 -> C
Если (ACC) = (ACCB)
To 1 -> C

Влияет на C.

ОПИСАНИЕ Содержимое аккумулятора (ACC) сравнивается содержимым буфера аккумулятора (ACCB). Большее значение (знаковое) загружается в оба регистра. Бит переноса изменяется в зависимости от результата сравнения. Если содержимое аккумулятора равно содержимому буфера аккумулятора, то в бит переноса заносится 1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 CRGT

До выполнения		После выполнения	
ACCB	4h	ACCB	5h
ACC	5h	ACC	5h
C	0	C	1

ПРИМЕР 2 CRGT

До выполнения		После выполнения	
ACCB	5h	ACCB	5h
ACC	5h	ACC	5h
C	0	C	1

CRLT - Тестирование ACC < ACCB

СИТАКСИС CRLT

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
Если (ACC) < (ACCB)
To (ACC) -> (ACCB); 1 -> C
Если (ACC) > (ACCB)
To (ACCB) -> (ACC); 0 -> C
Если (ACC) = (ACCB)
To 0 -> C

Влияет на C.

ОПИСАНИЕ Содержимое аккумулятора (ACC) сравнивается содержимым буфера аккумулятора (ACCB). Меньшее значение (знаковое) загружается в оба регистра. Бит переноса изменяется в зависимости от результата сравнения. Если содержимое аккумулятора равно содержимому буфера аккумулятора, то в бит переноса заносится 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
CRLT
До выполнения                После выполнения
ACCB    5h                   ACCB    4h
ACC     4h                   ACC     4h
C       0                    C       1
```

ПРИМЕР 2

```
CRLT
До выполнения                После выполнения
ACCB    4h                   ACCB    4h
ACC     4h                   ACC     4h
C       1                    C       0
```

DMOV – Перемещение данных в памяти данных

СИТАКСИС Прямая: DMOV dma
Косвенная: DMOV {ind}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(dma) -> dma + 1

Зависит от CNF и OVLY.

ОПИСАНИЕ Содержимое заданного адреса (dma) памяти данных копируется в следующий (dma+1) адрес памяти данных. DMOV выполнятся только для блоков внутрикристальной памяти конфигурируемой как данные. Как дополнение, инструкция продолжит перемещение данных за границы адресного пространства блоков ДОЗУ В0 и В1. Инструкция DMOV не может быть использована для внешней памяти или регистров, картированных в памяти в памяти. Если она будет использована для внешней памяти или регистров, картированных в памяти, то эффект выполнения инструрукции не будет достигнут. Когда данные копируются из адресуемой ячейки в следующую ячейку, содержание адресуемой ячейки остается неизменной.
Инструкция перемещения данных полезна для выполнения z^{-1} задержки, которые встречаются в цифровой обработке сигналов. Инструкция DMOV включена в команды LTD, MACD, MADD.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n-2	2n-2	2n-2, 2n+1†	2n-2+p
External	4n-2+2nd	4n-2+2nd	4n-2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
DMOV  DAT8  ; (DP = 6)
```

До выполнения		После выполнения	
Память данных		Память данных	
308h	43h	308h	43h
Память данных		Память данных	
309h	2h	309h	43h

ПРИМЕР 2

```
DMOV   *,AR1
```

До выполнения		После выполнения	
ARP	0	ARP	1
AR1	30Ah	AR1	30Ah
Память данных		Память данных	
30Ah	40h	30Ah	40h
Память данных		Память данных	
30Bh	41h	30Bh	40h

EXAR – Обмен значений буффера аккумулятора с аккумулятором

СИТАКСИС EXAR

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	0	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACCB) <-> (ACC)

ОПИСАНИЕ Содержимое аккумулятора обменивается с содержимым буфера аккумулятора (ACCB).

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР EXAR

До выполнения		После выполнения	
ACCB	02h	ACCB	043h
ACC	043h	ACC	02h

IDLE - Ожидание прерывания

СИТАКСИС IDLE

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	1	0

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC

Зависит от INTM.

ОПИСАНИЕ Инструкция IDLE обеспечивает режим ожидания выполнения последовательности инструкций до появления немаскируемого прерывания (внешнее или внутренне) PC увеличивается на 1 и процессор останавливается в состоянии ожидания до появления прерывания. Процессор выводится из состояния ожидания немаскируемым прерыванием даже, если INTM = 1. Если INTM = 1, то программа продолжит выполнение с команды, следующей за IDLE. Если INTM = 0, то вызывается программа обработки прерывания. Исполнение IDLE заставляет 1867ВЦ2Т входить в режим низкого потребления энергии.
В течение режима ожидания по IDLE таймер и Последовательный порт активны. Поэтому прерывания таймера и последовательного порта выводит процессор из этого режима.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР
```
IDLE  ; Процессор ожидает сброс или немаскируемое
        прерывание.
```

IDLE2 – Ожидание прерывания – режим малого потребления энергии

СИТАКСИС IDLE2

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	0	0	0	1	1

ОПЕРАНДЫ Нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC

Зависит от INTM.

ОПИСАНИЕ Инструкция IDLE2 обеспечивает режим ожидания выполнения последовательности инструкций до появления немаскируемого прерывания (внешнее или внутренне) PC увеличивается на 1 и процессор останавливается в состоянии ожидания до появления прерывания. Функциональный тактирующий сигнал выключается, что обеспечивает режима экстремально-низкого потребления энергии. PC
Процессор выводится из состояния низкого потребления энергии немаскируемым прерыванием, даже если на INTM=1. Если на INTM высокий уровень, то программа продолжит выполнение с команды следующей за IDLE2. Если на INTM низкий уровень, то вызывается программа обработки прерывания. Исполнение IDLE2 заставляет 1867ВЦ2Т (1867ВЦ2АТ) входить в режим низкого потребления энергии. Отличие от IDLE режима – это то, что IDLE2 режим не имеет активной периферии (последовательный порт или таймер).

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР
```
IDLE2  ; Процессор бездействует до сброса или немас-
         кируемого прерывания.
```

IN – Считывание данных из порта

СИТАКСИС Прямая : IN dma, PA
Косвенная : IN {ind}, PA[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ PA ≤ 65535

ИСПОЛНЕНИЕ
(PC) + 2 -> PC
Адрес порта -> адресная шины A15-A0
Шина данных D15-D0 -> dma

ОПИСАНИЕ Команда IN читает 16-ти битовое значение из внешнего порта ввода/вывода в заданную ячейку памяти данных. Сигнал ~IS устанавливается в 0, что обеспечивает выбор пространства ввода/вывода. Сигналы ~STRB,~RD и READY устанавливаются как при чтении внешней памяти данных. Заметьте, что адреса портов PA0-PA15 картируются в памяти, а других портов – нет.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM	$2+io_{src}$	$2+io_{src}$	$2+io_{src}$	$3+io_{src}+2p_{code}$
Destination:SARAM	$2+io_{src}$	$2+io_{src}$	$2+io_{src}$, $3+io_{src}$†	$3+io_{src}+2p_{code}$
Destination:External	$3+d_{dst}+io_{src}$	$3+d_{dst}+io_{src}$	$3+d_{dst}+io_{src}$	$6+d_{dst}+io_{src}+2p_{code}$

† If the operand and the code are in the same SARAM block

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$	$2n+1+nio_{src}+2p_{code}$
Destination: SARAM	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$, $2n+2+nio_{src}$†	$2n+1+nio_{src}+2p_{code}$
Destination: External	$4n-1+nd_{dst}+nio_{src}$	$4n-1+nd_{dst}+nio_{src}$	$4n-1+nd_{dst}+nio_{src}$	$4n+2+nd_{dst}+nio_{src}+2p_{code}$

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
IN  DAT7, PA5  ;Чтение слова из адреса 5 переферийно-
                го порта. Значение запоминается в
                307h ячейки памяти (DP=6).
```

```
ПРИМЕР 2        IN  *,PA0       ;Чтение слова из адреса 0 переферийно-
                                 го порта. Значение запоминается в
                                 ячейки памяти указанной текущим до-
                                 полнительным регистром.
```

INTR – Программное прерывание

СИТАКСИС INTR k

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	1	INTR#†				

ОПЕРАНДЫ $0 \le k \le 31$

ИСПОЛНЕНИЕ (PC) + 1 -> стек
соответствующий вектор прерывания -> PC

Не зависит от INTM.

ОПИСАНИЕ Инструкция INTM – программное прерывание, которое передает программный контроль адресу программной памяти заданный k (смотри следующую таблицу). Эта команда позволяет исполнять любую сервисную программу прерывания из программного обеспечения пользователя. Во время исполнения команды содержание PC + 1 заносится в стек. Заметьте, что программные прерывания INTR не маскируются. INTR для внешних прерываний(INT1-INT4) и выглядит точно так же, как внешнее прерывание (генерируется сигнал подтверждения прерывания и очищается соответствующий бит IFR).

K	Прерывания	Ячейки	K	Прерывания	Ячейки
0	$\overline{\text{RS}}$	0h	16	Резервное	20h
1	$\overline{\text{INT1}}$	2h	17	TRAP	22h
2	$\overline{\text{INT2}}$	4h	18	$\overline{\text{NMI}}$	24h
3	$\overline{\text{INT3}}$	6h	19	Резервное	26h
4	TINT	8h	20	Пользовательское	28h
5	RINT	Ah	21	Пользовательское	2Ah
6	XINT	Ch	22	Пользовательское	2Ch
7	TRNT	Eh	23	Пользовательское	2Eh
8	TXNT	10h	24	Пользовательское	30h
9	$\overline{\text{INT4}}$	12h	25	Пользовательское	32h
10	Резервное	14h	26	Пользовательское	34h
11	Резервное	16h	27	Пользовательское	36h
12	Резервное	18h	28	Пользовательское	38h
13	Резервное	1Ah	29	Пользовательское	3Ah
14	Резервное	1Ch	30	Пользовательское	3Ch
15	Резервное	1Eh	31	Пользовательское	3Eh

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

ПРИМЕР
```
INTR   3  ;Контроль передается ячейке 6h программной
          памяти, PC + 1 заносится в стек.
```

LACB – Загрузить ACCB в ACC

СИНТАКСИС [метка] LACB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACCB) -> ACC

ОПИСАНИЕ В аккумулятор загружается содержимое аккумуляторного буфера(ACCB).

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР

```
LACB
До выполнения          После выполнения
ACC     1376h          ACC     5555AAAAh
ACCB    5555AAAAh      ACCB    5555AAAAh
```

LACC – Загрузить ACC со сдвигом

СИНТАКСИС

Прямая	LACC dma [,сдвиг]
Косвенная	LACC {ind} [,сдвиг[,новый ARP]]
Непосредственная	LACC #lk [,сдвиг]

КОД КОМАНДЫ

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1				SHFT†	0							dma

†See Table 6–1 on page 6-2.

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1				SHFT†	1					See Section 5.2		

†See Table 6–1 on page 6-2.

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0							dma

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	1					See Section 5.2		

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	0	0				SHFT†
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ сдвиг ≤ 15 (по умолчанию 0)
-32768 ≤ lk ≤ 32768

ВЫПОЛНЕНИЕ

Прямая или косвенная адресация:
(PC) + 1 -> PC
(dma) * $2^{\text{сдвиг}}$ ->ACC

Длинная непосредственная адресация:
(PC) + 2 -> PC
lk * $2^{\text{сдвиг}}$ ->ACC

Зависит от SXM

ОПИСАНИЕ

Содержимое указанного адреса памяти данных или 16-битная константа сдвигается влево и загружается в аккумулятор. При сдвигании младшие биты заполняются 0. В старших битах происходит расширение знака, если SXM = 1, иначе они заполняются 0.

СЛОВА

1 (Прямая или косвенная адресация)
2 (Длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
LACC    DAT6, 4 ; (DP = 8,  SXM = 0)

До выполнения           После выполнения
Память данных           Память данных
406h    01h             406h    01h
ACC     012345678h C=X  ACC     10h C=X
```

ПРИМЕР 2

```
LACC    *, 4 ; (SXM = 0)
До выполнения           После выполнения
ARP     2               ARP     2
AR2     0300h           AR2     0300h
Память данных           Память данных
300h    0ffh            300h    0ffh
ACC     012345678h;C=X  ACC     0ffh; C=X
```

ПРИМЕР 3

```
LACC    #f000h, 1 ; (SXM = 1)
До выполнения           После выполнения
ACC     012345678h;C=X  ACC     0ffffe000h; C=X
```

LACL – Загрузить младшее слово и очистить старшее слово аккумулятора

СИНТАКСИС

Прямая	LACL dma
Косвенная	LACL {ind} [,новый ARP]
Непосредственная	LACL #k

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	1	See Section 5.2							

ОПЕРАНДЫ

$0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7
$0 \le k \le 255$

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC

Прямая или косвенная адресация:
0 -> ACC(31-16)
dma - >ACC(15-0)

Длинная непосредственная адресация:
0 -> ACC(31-16)
k - >ACC(15-0)

Не зависит от SXM

ОПИСАНИЕ

Содержимое указанного адреса памяти данных
или дополненная 0 в старших разрядах 8-бит.
константа загружается в младшее слово аккумулятора.
Старшее слово аккумулятора заполняется 0.
Данные интерпретируются как беззнаковые.
Расширение знака не выполняется независимо
от SXM.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory

DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР 1
```
LACL    DAT1 ; (DP = 6)
До выполнения           После выполнения
Память данных           Память данных
301h    01h             301h    01h

ACC     7fffffffh; C=X  ACC     0h; C=X
```

ПРИМЕР 2
```
LACC    *-, AR4
До выполнения           После выполнения
ARP     0               ARP     4
AR0     401h            AR20    400h
Память данных           Память данных
401h    0ffh            401h    0ffh
ACC     7fffffffh; C=X  ACC     0ffh; C=X
```

ПРИМЕР 3
```
LACC    #10h
До выполнения           После выполнения
ACC     7fffffffh; C=X  ACC     010h; C=X
```

LACT – Загрузить ACC со сдвигом, заданным в TREG1

СИНТАКСИС

Прямая	LACT dma
Косвенная	LACT {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
dma << TREG1(3-0) -> ACC
if(SXM == 1)
     Расширение знака dma;
else
     Нет расширение знака dma;
```

Зависит от SXM.

ОПИСАНИЕ

LACT загружает аккумулятор сдвинутым влево значением из памяти данных. Величина сдвига задана 4 младшими битами TREG1 (сдвиг на 1-15 разрядов). В старших битах происходит расширение знака если SXM = 1, иначе они заполняются 0.

LACT может использоваться для денормализации числа с плавающей точкой, если экспонента хранится в 4 младших битах регистра TREG1, а мантисса считывается из памяти. Этот способ денормализации может использоваться, когда размер экспоненты 4 бита или меньше.

Программная совместимость с 1867ВМ2 обеспечивается установкой бита TRM в регистре PMST в 0. В этом случае загружаемое в TREG0 значение пишется во все остальные TREG. Последующий вызов LACT получает из TREG1 верное значение. Этим обеспечивается совместимость на уровне объектных кодов.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LACT    DAT1 ; (DP = 6, SXM = 0)
До выполнения           После выполнения
Память данных           Память данных
301h    1376h           301h    1376h
ACC     98f7ec83h C=X   ACC     13760h C=X
TREG1   14h             TREG1   14h
```

ПРИМЕР 2

```
LACC    *-, AR3 ; (SXM = 1)
До выполнения           После выполнения
ARP     1               ARP     3
AR0     310h            AR20    300h
Память данных           Память данных
310h    0ff00h          310h    0ff00h
ACC     98f7ec83h C=X   ACC     0fffffe00h C=X
TREG1   11h             TREG1   11h
```

LAMM – Загрузить в аккумулятор отображаемый в память регистр

СИНТАКСИС

Прямая	LAMM dma
Косвенная	LAMM {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	Data Memory Address						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$

$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
dma -> ACC(15-0)
0 -> ACC(31-16)
```

ОПИСАНИЕ

В младшее слово аккумулятора загружается содержимое отображаемого в память регистра. 9 старших битов адреса данных обнуляются независимо от текущего значения DP или старших 9 битов текущего AR. Эта инструкция позволяет загрузить в аккумулятор значение любой ячейки памяти данных в странице 0 без модификации поля DP в регистре статуса ST0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Execution

Operand	ROM	DARAM	SARAM	External Memory
MMRT†	1	1	1	1+p
MMPORT	$1+io_{src}$	$1+io_{src}$	$1+iod_{src}$	$1+2+p+iod_{src}$

† Add one more cycle for peripheral memory-mapped access

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
MMRT‡	n	n	n	n+p
MMPORT	$n+mio_{src}$	$n+mio_{src}$	$n+mio_{src}$	$n+p+mio_{src}$

‡ Add n more cycle for peripheral memory-mapped access

ПРИМЕР 1

```
LAMM    BMAR ; (DP = 6)
До выполнения          После выполнения
ACC     22221376h      ACC     5555h
BMAR    5555h          BMAR    5555h
Память данных          Память данных
31fh    1000h          31fh    1000h
```

ПРИМЕР 2

```
LAMM    *
```

До выполнения		После выполнения	
ARP	1	ARP	1
AR0	325h	AR20	325h
ACC	22221376h	ACC	0fh
PRD	0fh	PRD	0fh
Память данных		Память данных	
325h	1000h	325h	1000h

Обратите внимание, что значение ячейки памяти 325h не загружается в аккумулятор. Загружается значение ячейки 25h.

LAR – Загрузить дополнительный регистр

СИНТАКСИС

Прямая	LAR ARx, dma
Косвенная	LAR ARx, {ind} [,новый ARP]
Короткая непосредственная	LAR ARx, #k
Длинная непосредственная	LAR ARx, #lk

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0		ARX†		0	dma						

†See Table 6–1 on page 6-2.

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0		ARX†		1	See Section 5.2						

†See Table 6–1 on page 6-2.

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0		ARX†		8-Bit Constant							

†See Table 6–1 on page 6-2.

Long immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	0	1		ARX†	
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ ARx ≤ 7
0 ≤ новый ARP ≤ 7
0 ≤ k ≤ 255
0 ≤ lk ≤ 65535

ВЫПОЛНЕНИЕ

Прямая или косвенная адресация:
(PC) + 1 -> PC
(dma) -> ARx

Короткая непосредственная адресация:
(PC) + 1 -> PC
k -> ARx

Длинная непосредственная адресация:
(PC) + 2 -> PC
lk -> ARx

ОПИСАНИЕ

Содержимое заданного адреса памяти или 8- или 16-битная константы загружается в заданный дополнительный регистр. Константы интерпретируются как беззнаковые число независимо от SXM.

LAR и SAR могут быть использованы для загрузки и сохранения дополнительных регистров во время вызовов подпрограмм или прерываний. Если дополнительный регистр не будет использоваться для косвенной адресации, LAR и SAR позволяют использовать его как дополнительный регистр хранения данных, особенно для обмена значениями ячеек памяти без изменения содержимого ACC.

СЛОВА 1 (прямая, косвенная короткая непосредственная адресация)
2 (длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
LAR    AR0, DAT16 ; (DP = 6)
До выполнения          После выполнения
Память данных          Память данных
310h    18h            310h    18h
AR0     6h             AR0     18h
```

ПРИМЕР 2
```
LAR    AR4, *-  ; (ARP = 4)
До выполнения          После выполнения
Память данных          Память данных
300h    32h            300h    32h
AR4     300h           AR4     32h
```

ПРИМЕР 3
```
LAR    AR4, #01h
До выполнения          После выполнения
AR4     0ff09h         AR4     01h
```

ПРИМЕР 4
```
LAR    AR4, #3fffh
До выполнения          После выполнения
AR4     0h             AR4     3fffh
```

LDP – Загрузить указатель страницы данных

СИНТАКСИС

Прямая	LDP dma
Косвенная	LDP {ind} [,новый ARP]
Короткая непосредственная	LDP #k

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	9-Bit Constant								

ОПЕРАНДЫ

$0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7
$0 \le k \le 511$

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC

Прямая или косвенная адресация:

9 младших битов (dma) -> DP

Короткая непосредственная адресация:

k -> DP

ОПИСАНИЕ

9 младших битов содержимого памяти или 9-битный непосредственный операнд загружается в регистр DP. DP конкатенируется с 7-битным адресом памяти данных для получения 16-битного адреса. DP может также загружаться командой LST.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

ПРИМЕР 1

```
LDP    DAT127 ; (DP = 511)
До выполнения          После выполнения
Память данных          Память данных
0ffffh  0fedch         0ffffh  0fedch
DP      1ffh           DP      0dch
```

ПРИМЕР 2

```
LDP    #0h
До выполнения          После выполнения
DP      1ffh           DP      0h
```

ПРИМЕР 3

```
LDP    *, AR5
До выполнения          После выполнения
ARP     4              ARP     5
AR4     300h           AR4     300h
Память данных          Память данных
300h    06h            300h    06h
DP      1ffh           DP      06h
```

LMMR – Загрузить отображаемый в память регистр

СИНТАКСИС

Прямая	LMMR dma, #lk
Косвенная	LMMR {ind}, #lk [,новый ARP]

КОД КЛМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

$0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7
$0 \le lk \le 65535$

ВЫПОЛНЕНИЕ

PFC -> MSC
(PC) + 2 -> PC
lk -> PFC
(src, адресуемый через PFC) ->(регистр, адресуемый младшими 7 битами dma)

ОПИСАНИЕ

В отображаемый в память регистр, указываемый 7 младшими битами dma, загружается длинная константа. 9 старших битов адреса pdma обнуляются независимо от текущего значения DP или старших 9 битов ARx. Эта инструкция позволяет загрузить любую ячейку памяти данных в странице 0 без модификации поля DP в регистре статуса ST0.

СЛОВА 2

ЦИКЛЫ

ПРИМЕР 1

```
LMMR    DMBR, #300h
До выполнения          После выполнения
Память данных          Память данных
300h    1376h          300h    1376h
DMBR    5555h          DMBR    300h
```

ПРИМЕР 2

```
LMMR    *, #300h, AR4 ; &CBCR = 1eh
До выполнения          После выполнения
ARP     0              ARP     14
AR0     31eh           AR20    31eh
Память данных          Память данных
300h    20h            300h    20h
CBCR    0h             CBCR    20h
```

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: MMR‡	2	2	2	$2+2p_{code}$

Source: SARAM Destination: MMR‡	2	2	2, 3†	$2+2p_{code}$
Source: External Destination: MMR‡	$2+p_{src}$	$2+p_{src}$	$2+p_{src}$	$3+p_{src}+2p_{code}$
Source: DARAM Destination: MMPORT	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$	$5+2p_{code}+io_{dst}$
Source: SARAM Destination: MMPORT	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$, 4†	$5+2p_{code}+io_{dst}$
Source: External Destination: MMPORT	$3+p_{src}+io_{dst}$	$3+p_{src}+io_{dst}$	$3+p_{src}+io_{dst}$	$6+p_{src}+2p_{code}+io_{dst}$

† If the source operand and the code are in the same SARAM block
‡ Add one more cycle for peripheral memory-mapped register access

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: MMR§	2n	2n	2n	$2n+2p_{code}$
Source: SARAM Destination: MMR§	2n	2n	2n, 2n+1†	$2n+2p_{code}$
Source: External Destination: MMR§	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}+2p_{code}$
Source: DARAM Destination: MMPORT	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}+2p_{code}$
Source: SARAM Destination: MMPORT	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$, $3n+1+nio_{dst}$ †	$3n+nio_{dst}+2p_{code}$
Source: External Destination: MMPORT	$4n-1+nd_{src}+nio_{dst}$	$4n-1+nd_{src}+nio_{dst}$	$4n-1+nd_{src}+nio_{dst}$	$4n+2+nd_{src}+nio_{dst}+2p_{code}$

† If the source operand and the code are in the same SARAM block
§ Add n more cycle for peripheral memory-mapped register access

LPH – Загрузить старшее слово регистра P

СИНТАКСИС Прямая LPH dma
Косвенная LPH {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	1	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(dma) -> регистр P(31-16)

ОПИСАНИЕ В старшее слово регистра P загружается содержимое памяти данных. Младшее слово P не меняется.

LPH может использоваться для восстановления P после прерывания или вызова подпрограммы.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
LPH    DAT0 ; (DP = 4)
До выполнения          После выполнения
Память данных          Память данных
200h    0f79ch         200h    0f79ch
P       30079844h      P       0f79c9844h
```

ПРИМЕР 2
```
LPH    *, AR6
До выполнения          После выполнения
ARP     5              ARP     6
AR5     200h           AR5     200h
Память данных          Память данных
200h    0f79ch         200h    0f79ch
P       30079844h      P       0f79c9844h
```

LST – Загрузить регистр статуса

СИНТАКСИС

Прямая	LST #n, dma
Косвенная	LST #n, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing for LST #0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	0	dma						

Indirect addressing for LST #0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	1	See Section 5.2						

Direct addressing for LST #1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	dma						

Indirect addressing for LST #1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$
$0 \le n \le 1$
$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC
(dma) -> регистр статуса STn
dma (биты 13-15) -> ARP(независимо от новый ARP)

Влияет на ARB, ARP, OV, OVM, DP, CNF, TC, SXM, C, HM, XF, PM.
Не влияет на INTM.

ОПИСАНИЕ

В регистр статуса STn загружается значение из памяти данных. Обратите внимание, что бит INTM не затрагивается LST #0. Кроме того, LST #0 не влияет на поле ARB в ST1 даже, если загружается новое ARP. Если новое значение ARP заданно в косвенном режиме адресации, оно игнорируется, а вместо него в ARP загружается значение из адреса памяти данных.

Примечание. Когда загружается ST1, значение, загружаемое в ARB, также загружается в ARP.

LST может использоваться для восстановления ST0, ST1 после прерывания или вызова подпрограммы.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2	2	2	$2+p_{code}$
Source: SARAM	2	2	2, 3†	$2+p_{code}$
Source: External	$2+d_{src}$	$2+d_{src}$	$2+d_{src}$	$3+d_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	2n	2n	2n	$2n+p_{code}$
Source: SARAM	2n	2n	2n, 2n+1†	$2n+p_{code}$
Source: External	$2n+nd_{src}$	$2n+nd_{src}$	$2n+nd_{src}$	$2n+1+nd_{src}+p_{code}$

† If the source operand and the code are in the same SARAM block

ПРИМЕР 1

```
MAR     *, AR0
LST    #0, *, AR1 ; Содержимое памяти данных,
 ; адресуемой AR0, загружается в ST0, исключая
 ; бит INTM. Даже когда задано новое значение
 ; ARP, оно игнорируется и старое ARP не загружается
 ; в ARB
```

ПРИМЕР 2

```
LST     #0, 60h ; (DP = 0)
До выполнения           После выполнения
Память данных           Память данных
60h     2404h           60h     2404h
ST0     6e00h           ST0     2604h
ST1     0580h           ST1     0580h
```

ПРИМЕР 3

```
LST     #0, *-, AR1
До выполнения           После выполнения
ARP     4               ARP     1
AR4     3ffh            AR4     2feh
Память данных           Память данных
3ffh    0ee04h          3ffh    0ee04h
ST0     0ee00h          ST0     0ee04h
ST1     f780h           ST1     f780h
```

ПРИМЕР 4

```
LST     #1, 0h  ;(DP = 6)
До выполнения           После выполнения
Память данных           Память данных
300h    0e1bch          300h    0e1bch
ST0     0406h           ST0     0e406h
ST1     09a0h           ST1     0e1bch
```

LT – Загрузить TREG0

СИНТАКСИС

Прямая	[метка] LT dma
Косвенная	[метка] LT {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
(dma) -> TREG0
if(TRM == 0){
   (dma) -> TREG1;
   (dma) -> TREG2;
}
```

Зависит от TRM.

ОПИСАНИЕ

В TREG0 загружается значение из памяти данных (dma). LT может использоваться для загрузки TREG0 при подготовке к умножению (см. LTA, LTD, LTP, LTS, MPY, MPYA, MPYS, MPYU). Программная совместимость с 1867ВМ2 обеспечивается установкой бита TRM в регистре PMST в 0. В этом случае загружаемое в TREG0 значение пишется во все остальные TREG. TREG – отображаемые в память регистры, они могут быть записаны или прочитаны любой инструкцией доступа к памяти.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LT    DAT24 ; (DP = 8, TRM = 1)
До выполнения          После выполнения
Память данных          Память данных
418h    62ch           418h    62ch
TREG0   3h             TREG0   62h
```

ПРИМЕР 2

```
LT    *, AR3 ; (TRM = 0)
До выполнения          После выполнения
ARP     2              ARP    3
AR2     418h           AR2 418h
Память данных          Память данных
418h    62ch           418h    62ch
TREG0   3h             TREG0   62h
TREG1   4h             TREG0   62h
TREG02  5h             TREG0   62h
```

LTA – Загрузить TREG0 и аккумулировать предыдущий результат

СИНТАКСИС
Прямая LTA dma
Косвенная LTA {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(dma) -> TREG0
(ACC) + сдвинутый регистр P -> ACC

Зависит от OVM, PM и TRM; влияет на OV и C.

ОПИСАНИЕ
В TREG0 загружается значение из памяти данных (dma). Содержимое регистра P, сдвинутое как определено битом стстуса PM, прибавляется к ACC. Есле TRM = 0, загружаемое в TREG0 значение пишется во все остальные TREG для совместимости с 1867ВМ2.

Функции LTA включаются в LTD.

СЛОВА
1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LTA    DAT36 ; (DP = 6, PM = 0, TRM = 1)
До выполнения          После выполнения
Память данных          Память данных
324h    62h            324h    62h
TREG0   3h             TREG0   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     14h C=0
```

ПРИМЕР 2

```
LTA    *, AR5 ; (TRM = 0)
До выполнения          После выполнения
ARP     4              ARP     5
AR2     324h           AR2     324h
Память данных          Память данных
324h    62h            324h    62h
TREG0   3h             TREG0   62h
TREG1   4h             TREG0   62h
TREG02  5h             TREG0   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     14h C=0
```

LTD – Загрузить TREG0, аккумулировать предыдущий результат и переместить данные

СИНТАКСИС

Прямая LTD dma
Косвенная LTD {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC
(dma) -> TREG0
(dma) -> (dma + 1)
(ACC) + сдвинутый регистр P -> ACC

Зависит от OVM, PM и TRM; влияет на OV и C.

ОПИСАНИЕ

В TREG0 загружается значение из памяти данных (dma). Содержимое регистра P, сдвинутое как определено битом статуса PM, прибавляется к ACC. Содержимое указанной ячейки памяти копируется по следующему более старшему адресу. Если TRM = 0, загружаемое в TREG0 значение пишется во все остальные TREG для совместимости с 1867ВМ2.

Эта инструкция допустима для всех блоков ОЗУ, сконфигурированных как память данных. Функция перемещения данных переходит через границы непрерывных блоков данных, но не может быть использована с внешней памятью данных или отображаемыми в память регистрами. Эта функция описана в описании инструкции DMOV. Обратите внимание, что при работе с внешней памятью функция LTD идентична LTA.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n-2	2n-2	2n-2, 2n+1†	2n-2+p
External	4n-2+2nd	4n-2+2nd	4n-2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LTD    DAT126 ; (DP = 7, PM = 0, TRM = 1)
До выполнения          После выполнения
Память данных          Память данных
3feh    62h            3feh    62h
3ffh    0h             3fh     62h
TREG0   3h             TREG0   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     14h C=0
```

ПРИМЕР 2

```
LTD     *, AR3
До выполнения          После выполнения
ARP     1              ARP     3
AR1     3feh           AR1     3feh
Память данных          Память данных
3feh    62h            3feh    62h
3ffh    0h             3fh     62h
TREG0   3h             TREG0   62h
TREG1   4h             TREG1   62h
TREG2   5h             TREG2   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     14h C=0
```

LTP – Загрузить TREG0 и записать P в аккумулятор

СИНТАКСИС
Прямая [метка] LTP dma
Косвенная [метка] LTP {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(dma) -> TREG0
сдвинутый регистр P -> ACC

Зависит от PM и TRM

ОПИСАНИЕ
В TREG0 загружается значение из памяти данных (dma). Содержимое регистра P, сдвинутое как определено битом статуса PM, загружается в ACC. Если TRM = 0, загружаемое в TREG0 значение пишется во все остальные TREG для совместимости с 1867ВМ2.

СЛОВА
1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LTD    DAT36 ; (DP = 6, PM = 0, TRM = 1)
```

До выполнения		После выполнения	
Память данных		Память данных	
324h	62h	324h	62h
324h	0h	324h	62h
TREG0	3h	TREG0	62h
P	0fh	P	0fh
ACC	5h C=X	ACC	0fh C=X

ПРИМЕР 2

```
LTD     *, AR5 ; (PM = 0, TRM = 0)
До выполнения           После выполнения
ARP     2               ARP     5
AR1     324h            AR1     324h

Память данных           Память данных
324h    62h             324h    62h
3ffh    0h              3fh     62h
TREG0   3h              TREG0   62h
TREG1   4h              TREG1   62h
TREG2   5h              TREG2   62h
P       0fh             P       0fh
ACC     5h C=X          ACC     14h C=X
```

LTS – загрузить TREG0 и вычесть предыдущий результат

СИНТАКСИС

Прямая [метка] LTS dma

Косвенная [метка] LTS {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127

0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC

(dma) -> TREG0

(ACC) – (сдвинутый регистр P) -> ACC

Зависит от OVM, PM и TRM; влияет на OV и C.

ОПИСАНИЕ

В TREG0 загружается значение из памяти данных (dma). Содержимое регистра P, сдвинутое как определено битом статуса PM, вычитается из ACC. Если TRM = 0, загружаемое в TREG0 значение пишется во все остальные TREG для совместимости с 1867ВМ2.

Функции LTA включаются в LTD.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1,2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
LTS    DAT36 ; (DP = 6, PM = 0, TRM = 1)
До выполнения          После выполнения
Память данных          Память данных
324h    62h            324h    62h
TREG0   3h             TREG0   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     0FFFFFFF6h C=0
```

ПРИМЕР 2

```
LTS    *, AR2 ; (TRM = 0)
До выполнения          После выполнения
ARP     1              ARP     2
AR2     324h           AR2     324h
Память данных          Память данных
324h    62ch           324h    62ch
TREG0   3h             TREG0   62h
TREG1   4h             TREG0   62h
TREG02  5h             TREG0   62h
P       0fh            P       0fh
ACC     5h C=X         ACC     0FFFFFFF6h C=0
```

MAC – Умножить и аккумулировать

СИНТАКСИС

Прямая	MAC pma, dma
Косвенная	MAC pma, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ pma ≤ 65535
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 2 -> PC
(PFC) -> MCS
(pma) -> PFC

if((счетчик_повтора) != 0){
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать текущий AR и ARP как задано;
   (PFC) + 1 -> PFC;
   (счетчик_повтора) - 1 -> (счетчик_повтора);
}
else{
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать текущий AR и ARP как задано;
   (PFC) + 1 -> PFC;
}
(MSC) -> PFC;
```

Зависит от OVM, PM и TRM; влияет на OV и C.

ОПИСАНИЕ

Содержимое регистра P, сдвинутое как определено битом статуса PM, прибавляется к ACC.Инструкция MAC перемножает значение из памяти данных (заданное dma) со значением из памяти программ(заданным pma). Результат умножения сохраняется в P-регистре.

Адрес памяти данных и программ 1867ВЦ2Т, 1867ВЦ2АТ может быть любым не зарезервированным, внутри- и внекристальным. Если программная память – это блок B0 внутрикристального ОЗУ, бит CNF должен быть установлен в 1. Старшие 8 бит адреса программной памяти должны быть 0FFh для адресации блока B0 внутрикристального ОЗУ программ. Старшие 8 бит адреса памяти данных должны быть 0 для доступа к адресам ниже 500h. В прямом режиме адресации dma не может быть изменен во время повторения инструкции.

При повторении MAC адрес программной памяти, хранящийся в PFC, увеличивается на 1 во время операции, что позволяет обрабатывать серии операндов в памяти. MAC применяется

для вычисления длинных сумм произведений, т. к. выполняется как одноцикловая инструкция после старта конвейера PRT.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	3	3	3	$3^{+}2p_{code}$
1: SARAM 2: DARAM	3	3	3	$^{3+}2p_{code}$
1: External 2: DARAM	$3+p_{op1}$	$3^{+}p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	3	3	3	$3+2p_{code}$
1: SARAM 2: SARAM	3,4†	3,4†	3,4†	$3+2p_{code}$, $4+2p_{code}$†
1: External 2: SARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3^{+}p_{op1}+2p_{code}$
1: DARAM/ROM 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: SARAM 2: External	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: External 2: External	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}+2p_{code}$

† If both operands are in the same SARAM block.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: SARAM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: External 2: DARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: SARAM 2: SARAM	n+2, 2n+2†	n+2, 2n+2†	n+2, 2n+2†	$n+2+2p_{code}$, 2n+2†
1: External 2: SARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
1: DARAM/ROM	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$

	2:				
External					
1: SARAM					
	2:	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}$	$n+2+nd_{op2}+2p_{code}$
External					
	1:				
External		$2n+2+np_{op1}+nd_{op2}$	$2n+2+np_{op1}+nd_{op2}$	$2n+2+np_{op1}+nd_{op2}$	$2n+2+np_{op1}+nd_{op2}+2p_{code}$
	2:				
External					

† If both operands are in the same SARAM block.

ПРИМЕР 1

```
MAC    0FF000h, 02h ; (DP = 6, PM = 0, CNF = 1)
До выполнения          После выполнения
Память данных          Память данных
302h    23h            302h    23h
Память программ        Память программ
0ff00h  4h             0ff00h  4h
TREG0   45             TREG0   23h
P       458972h        P       08ch
ACC     723EC41h C=X   ACC     76975B3h C=0
```

ПРИМЕР 2

```
MAC    0FF00h, *, AR5 ; (PM = 0, CNF = 1)
До выполнения          После выполнения
ARP     4              ARP     5
AR2     302h           AR2     302h
302h    23h            302h    23h
Память программ        Память программ
0ff00h  4h             0ff00h  4h
TREG0   45             TREG0   23h
P       458972h        P       08ch
ACC     723EC41h C=X   ACC     76975B3h C=0
```

MACD – Умножить и аккумулировать С перемещением данных

СИНТАКСИС Прямая MACD pma, dma
Косвенная MACD pma, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	1	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ pma ≤ 65535
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 2 -> PC
(PFC) -> MCS
(pma) -> PFC

if((счетчик_повтора) != 0){
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать текущий AR и ARP как задано;
   (PFC) + 1 -> PFC;
   (dma) -> (dma + 1);
   (счетчик_повтора) - 1 -> (счетчик_повтора);
}
else{
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать текущий AR и ARP как задано;
   (PFC) + 1 -> PFC;
   (dma) -> (dma + 1);
}
(MSC) -> PFC;

Зависит от OVM, PM;  влияет на OV и C.
```

ОПИСАНИЕ Содержимое регистра P, сдвинутое как определено битами PM, прибавляется к ACC.Инструкция MACD перемножает значение из памяти данных (заданное dma) со значением из памяти программ(заданное pma).

Адрес памяти данных и программ может быть любым не зарезервированным, внутри- и внекристальным. Если программная память – это блок B0 внутрикристального ОЗУ, бит CNF должен быть установлен в 1. Старшие 8 бит адреса программной памяти должны быть 0FFh для адресации блока B0 внутрикристального ОЗУ программ. Старшие 8 бит адреса памяти данных должны быть 0 для доступа к адресам ниже 500h. В прямом режиме адресации dma не может быть изменен во время повторения инструкции. Если MACD адресует отображаемый в память регистр или внешнюю память, эффект от MACD такой же как от MAC (см. описание DMOV). MACD делает то же, что и MAC и дополнительно перемещает

данные во внутренней памяти. Это делает MACD полезным в таких применениях как вычисление свёрток.
При повторении MACD адрес программной памяти, хранящийся в PFC, увеличивается на 1 во время операции, что позволяет обрабатывать серии операндов в памяти. MACD применяется для вычисления длинных сумм произведений, т. к. выполняется как одноцикловая инструкция после старта в цикле PRT.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	3	3	3	$3^{+}2p_{Code}$
1: SARAM 2: DARAM	3	3	3	$3^{+}2p_{Code}$
1: External 2: DARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	3	3	3	$3^{+}2p_{Code}$
1: SARAM 2: SARAM	3	3	3, 4‡, 5§	$3+2p_{code}$, $4+2p_{code}$‡
1: External 2: SARAM	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}$	$3+p_{op1}+2p_{code}$
1: DARAM/ROM 2: External¶	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: SARAM 2: External¶	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}$	$3+d_{op2}+2p_{code}$
1: External 2: External¶	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}$	$4+p_{op1}+d_{op2}+2p_{code}$

‡ If both operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: SARAM 2: DARAM	n+2	n+2	n+2	$n+2+2p_{code}$
1: External 2: DARAM	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}$	$n+2+np_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	2n	2n	2n, 2n+2†	$2n+2p_{code}$

1: SARAM 2: SARAM	2n, 3n‡	2n, 3n‡	2n, 2n+2†, 3n‡,3n+2§	2n+2p_{code},3n‡
1: External 2: SARAM	2n+np_{op1}	2n+np_{op1}	2n+np_{op1}, 2n+2+np_{op1} †	2n+np_{op1}+2p_{code}
1: DARAM/ROM 2: External	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}+2p_{code}
1: SARAM 2: External	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}	n+2+nd_{op2}+2p_{code}
1: External 2: External	2n+2+np_{op1} +nd_{op2}	2n+2+np_{op1}+nd_{op2}	2n+2+np_{op1} +nd_{op2}	2n+2+np_{op1}+nd_{op2} +2p_{code}

† If both operand2 are in the same SARAM block.
‡ If both operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

ПРИМЕР 1

```
MACD    0FF000h, 02h ; (DP = 6, PM = 0, CNF = 1)
До выполнения           После выполнения
Память данных           Память данных
308h    23h             308h    23h
309h    18h             309h    23h
Память программ         Память программ
0ff00h  4h              0ff00h  4h
TREG0   45              TREG0   23h
P       458972h         P       08ch
ACC     723EC41h C=X    ACC     76975B3h C=0
```

ПРИМЕР 2

```
MACD    0FF00h, *, AR6 ; (PM = 0, CNF = 1)
До выполнения           После выполнения
ARP     5               ARP     6
AR2     308h            AR2     308h
Память  данных          Память данных
308h    23h             308h    23h
309h    18h             309h    23h
Память программ         Память программ
0ff00h  4h              0ff00h  4h
TREG0   45              TREG0   23h
P       458972h         P       08ch
ACC     723EC41h C=X    ACC     76975B3h C=0
```

Примечание. Функция перемещения данных MACD работает только с внутрикристальным ОЗУ.

MADD – Умножить и аккумулировать С перемещением данных и динамической адресацией

СИНТАКСИС

Прямая	MADD dma
Косвенная	MADD {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 2 -> PC
(PFC) -> MCS
(BMAR) -> PFC

if((Счетчик_повтора) != 0){
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать AR(ARP) и ARP как задано;
   (PFC) + 1 -> PFC;
   (dma) -> (dma + 1);
   (Счетчик_повтора) - 1 -> (Счетчик_повтора);
}
else{
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать AR(ARP) и ARP как задано;
   (PFC) + 1 -> PFC;
   (dma) -> (dma + 1);
}
(MSC) -> PFC;
```

Зависит от OVM, PM; влияет на OV и C.

ОПИСАНИЕ

Содержимое регистра P, сдвинутое как определено битами PM, прибавляется к ACC.Инструкция MADD перемножает значение из памяти данных(заданное dma) со значением из памяти программ(заданным pma). Адрес программной памяти содержится в регистре BMAR вместо указания его в виде длинного непосредственного операнда.
Адрес памяти данных и программ может быть любым незарезервированным, внутри- и внекристальным. Если программная память – это блок B0 внутрикристального ОЗУ, бит CNF должен быть установлен в 1. Старшие 8 бит адреса программной памяти должны быть 0FFh для адресации блока B0 внутрикристального ОЗУ программ. Старшие 8 бит адреса памяти данных должны быть 0 для доступа к адресам ниже 500h. В прямом режиме адресации dma не может быть изменен во время повторения инструкции. Если MADD адресует отображаемый в память регистр или внешнюю память, эффект от MADD такой же как от MADS (см. описание DMOV). MADD делает то же, что и MADS и дополнительно перемещает данные во внутренней памяти. Это делает MADD полезным

в таких применениях как вычисление свёрток.
При повторении MADD адрес программной памяти, хранящийся в PFC, увеличивается на 1 во время операции, что позволяет обрабатывать серии операндов в памяти. MADD применяется для вычисления длинных сумм произведений, т. к. выполняется как одноцикловая инструкция после старта в уикле PRT.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	2	2	2	$2+p_{code}$
1: SARAM 2: DARAM	2	2	2	$2+p_{code}$
1: External 2: DARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+2p_{code}$
1: DARAM/ROM 2: SARAM	2	2	2	$2+p_{code}$
1: SARAM 2: SARAM	2	2	2,3‡,4§	$2+p_{code}$, $3+p_{code}$‡
1: External 2: SARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+p_{code}$
1: DARAM/ROM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: SARAM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: External 2: External	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}+p_{code}$

‡ If both operands are in the same SARAM block
§ If both operands and the code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: SARAM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: External 2: DARAM	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}+p_{code}$
1: DARAM/ROM 2: SARAM	2n-1	2n-1	2n-1, 2n+1†	$2n-1+p_{code}$
1: SARAM 2: SARAM	2n-1, 3n-1‡	2n-1, 3n-1‡	2n-1, 2n+1†, 3n-1‡,3n+1§	$2n-1+p_{code}$,3n-1‡

1: External 2: SARAM	$2n-1+np_{op1}$	$2n-1+np_{op1}$	$2n-1+np_{op1}$, $2n+1+np_{op1}$ †	$2n-1+np_{op1}+p_{code}$
1: DARAM/ROM 2: External	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: SARAM 2: External	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: External 2: External	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}+p_{code}$

† If operand2 and code are in the same SARAM block
‡ If both operands are in the same SARAM block
§ If both operands and code are in the same SARAM block
¶ Data move operation is not performed when operand2 is in external data memory.

ПРИМЕР 1

```
MADD    DAT7 ; (DP = 6, PM = 0, CNF = 1)
До выполнения           После выполнения
Память данных           Память данных
307h    8h              307h    8h
309h    9h              308h    8h
BMAR    0ff00h          BMAR    0ff00h
TREG0   4eh             TREG0   8h
ff00h   2h              ff00h   2h
P       458972h         P       10h
ACC     723EC41h C=X    ACC     76975B3h C=0
```

ПРИМЕР 2

```
MADD    *, AR6 ; (PM = 0, CNF = 1)
До выполнения           После выполнения
ARP     5               ARP     6
AR2     308h            AR2     308h
Память  данных          Память данных
308h    23h             308h    23h
309h    18h             309h    23h
Память программ         Память программ
0ff00h  4h              0ff00h  4h
TREG0   45              TREG0   23h
P       458972h         P       08ch
ACC     723EC41h C=X    ACC     76975B3h C=0
```

MADS – Умножить и аккумулировать с динамической адресацией

СИНТАКСИС

Прямая	MADS dma
Косвенная	MADS {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

```
(PC) + 1-> PC
(PFC) -> MCS
(BMAR) -> PFC

if((счетчик_повтора) != 0){
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать AR(ARP) и ARP как задано;
   (PFC) + 1 -> PFC;
   (счетчик_повтора) - 1 -> (счетчик_повтора);
}
else{
   ACC + (сдвинутый регистр P) ->ACC;
   (dma) -> TREG0;
   (dma) * (pma, адресуемый PFC) -> P;
   модифицировать AR(ARP) и ARP как задано;
   (PFC) + 1 -> PFC;
}
(MSC) -> PFC;

Зависит от OVM, PM;  влияет на OV и C.
```

ОПИСАНИЕ

Содержимое регистра P, сдвинутое как определено битами PM, прибавляется к ACC.Инструкция MADS перемножает значение из памяти данных(заданное dma) со значением из памяти программ (заданным pma).
Адрес программной памяти содержится в регистре BMAR вместо указания его в виде длинного непосредственного операнда. Это обеспечивает динамическую адресацию таблиц коэффициентов. Адрес памяти данных и программ может быть любым незарезервированным, внутри- и внекристальным. Если программная память – это блок B0 внутрикристального ОЗУ, бит CNF должен быть установлен в 1. Старшие 8 бит адреса программной памяти должны быть 0FFh для адресации блока B0 внутрикристального ОЗУ программ. Старшие 8 бит адреса памяти данных должны быть 0 для доступа к адресам ниже 500h. В прямом режиме адресации dma не может быть изменен во время повторения инструкции.
При повторении MADS адрес программной памяти, хранящийся в PFC, увеличивается на 1 во время операции, что позволяет обрабатывать серии операндов в памяти. MADS применяется для вычисления длинных сумм произведений, т. к. выполняется как одно-цикловая инструкция после старта в цикле PRT.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	2	2	2	$2+p_{code}$
1: SARAM 2: DARAM	2	2	2	$2+p_{code}$
1: External 2: DARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	
1: DARAM/ROM 2: SARAM	2	2	2	$2+p_{code}$
1: SARAM 2: SARAM	2, 3†	2, 3†	2, 3†	$2+p_{code}$, $3+p_{code}$†
1: External 2: SARAM	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}$	$2+p_{op1}+p_{code}$
1: DARAM/ROM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: SARAM 2: External	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}$	$2+d_{op2}+p_{code}$
1: External 2: External	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}$	$3+p_{op1}+d_{op2}+p_{code}$

† If both operands are in the same SARAM block.

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
1: DARAM/ROM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: SARAM 2: DARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: External 2: DARAM	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}+p_{code}$
1: DARAM/ROM 2: SARAM	n+1	n+1	n+1	$n+1+p_{code}$
1: SARAM 2: SARAM	n+1, 2n+1†	n+1, 2n+1†	n+1, 2n+1†	$n+1+p_{code}$, 2n+1†
1: External 2: SARAM	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}$	$n+1+np_{op1}+p_{code}$
1: DARAM/ROM 2: External	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: SARAM 2: External	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}$	$n+1+nd_{op2}+p_{code}$
1: External 2: External	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}$	$2n+1+np_{op1}+nd_{op2}+p_{code}$

External

† If both operands are in the same SARAM block.

ПРИМЕР 1

```
MADS    DAT12 ; (DP = 6, PM = 0, CNF = 1)
До выполнения           После выполнения
Память данных           Память данных
30Ch    8h              30Ch    8h
BMAR    0ff00h          BMAR    0ff00h
TREG0   4eh             TREG0   8h
ff00h   2h              ff00h   2h
P       458972h         P       10h
ACC     723EC41h C=X    ACC     76975B3h C=0
```

ПРИМЕР 2

```
MADS    *, AR3 ; (PM = 0, CNF = 1)
До выполнения           После выполнения
ARP     5               ARP     6
AR2     308h            AR2     308h
Память  данных          Память данных
308h    23h             308h    23h
309h    18h             309h    23h
Память программ         Память программ
0ff00h  4h              0ff00h  4h
TREG0   45              TREG0   23h
P       458972h         P       08ch
ACC     723EC41h C=X    ACC     76975B3h C=0
```

MAR – модифицировать дополнительный регистр

СИНТАКСИС

```
Прямая          MAR  dma
Косвенная       MAR  {ind} [,новый ARP]
```

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1-> PC

Модифицирует ARP, AR как задано при косвенной адресации. Работает как NOP в режиме прямой адресации.

ОПИСАНИЕ

В режиме косвенной адресации модифицируется дополнительный регистр и ARP; ссылка на память никак не используется. Старое значение ARP копируется в поле ARB регистра статуса ST1. Любая операция, выполняемая MAR, может быть выполнена любой инструкцией, использующей косвенную адресацию. ARP также может быть загружен инструкцией LST.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
MAR    *, AR1 ; загрузка 1 в ARP
До выполнения          После выполнения
ARP     0              ARP     1
ARB     7              ARB     0
```

ПРИМЕР 1

```
MAR    *+, AR1 ; увеличить текущий дополнительный
               ; регистр и загрузить 1 в ARP
До выполнения          После выполнения
AR1     34h            AR1     35h
ARP     1              ARP     5
ARB     0              ARB     1
```

MPY – умножить

СИНТАКСИС

Прямая MPY dma
Косвенная MPY {ind} [,новый ARP]
Непосредственная MPY #k
Длинная непосредственная MPY #lk

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	0	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	13-Bit Constant												

Long immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
-4096 ≤ k ≤ 4095
-32768 ≤ lk ≤ 32767

ВЫПОЛНЕНИЕ

Прямая или косвенная адресация:
(PC) + 1 -> PC
(TREG0) * (dma) -> P

Короткая непосредственная адресация:
(PC) + 1 -> PC
(TREG0) * k -> P

Длинная непосредственная адресация:
(PC) + 2 -> PC
(TREG0) * lk -> P

ОПИСАНИЕ

Содержимое TREG0 умножается на содержимое ячейки памяти данных. Результат пишется в регистр P. При короткой непосредственной адресации TREG0 умножается на знаковую 13-битовую константу независимо от SXM.

СЛОВА

1 (Прямая, косвенная или короткая непосредственная адресация)
2 (Длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (short immediate addressing)

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
MPY    DAT13 ; (DP = 8)
До выполнения          После выполнения
Память данных          Память данных
40Dh    01h            40Dh    01h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
```

ПРИМЕР 2

```
MPY    *, AR2
До выполнения          После выполнения
ARP     1              ARP     2
AR2     40Dh           AR2     40Dh
Память данных          Память данных
40Dh    7h             40Dh    7h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
```

ПРИМЕР 3

```
MPY    #031h
До выполнения          После выполнения
TREG0   6h             TREG0   6h
P       36h            P       2Ah
```

ПРИМЕР 4

```
MPY    #01234h
До выполнения          После выполнения
TREG0   6h             TREG0   6h
P       36h            P       2468h
```

MPYA – Умножить и аккумулировать предыдущий результат

СИНТАКСИС Прямая MPYA dma
Косвенная MPYA {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC
(ACC) + (сдвинутый регистр P) -> ACC
(TREG0) * (dma) -> P

Зависит от OVM, PM; влияет на OV и C.

ОПИСАНИЕ Содержимое TREG0 умножается на содержимое ячейки памяти данных. Результат записывается в регистр P. Предыдущее значение P-регистра, сдвинутый, как определено битами PM, прибавляется к ACC.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
MPYA    DAT13 ; (DP = 6, PM = 0)
До выполнения          После выполнения
Память данных          Память данных
30Dh    23h            30Dh    23h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
ACC     54h C=X        ACC     8Ah C=0
```

ПРИМЕР 2

```
MPYA    *, AR4 ; (PM = 0)
До выполнения          После выполнения
ARP     3              ARP     4
AR3     30Dh           AR3     30Dh
Память данных          Память данных
30Dh    7h             30Dh    7h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
ACC     54h C=X        ACC     8Ah C=0
```

MPYS – Умножить и вычесть предыдущий результат

СИНТАКСИС

Прямая MPYS dma

Косвенная MPYS {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	1	See Section 5.2						

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ОПЕРАНДЫ

0 ≤ dma ≤ 127

0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC

(ACC) – (сдвинутый регистр P) -> ACC

(TREG0) * (dma) -> P

Зависит от OVM, PM; влияет на OV и C.

ОПИСАНИЕ

Содержимое TREG0 умножается на содержимое ячейки памяти данных. Результат записывается в регистр P. Предыдущее значение P-регистра, сдвинутый, как определено битами PM, прибавляется к ACC.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
MPYS    DAT13 ; (DP = 6, PM = 0)
До выполнения          После выполнения
Память данных          Память данных
30Dh    7h             30Dh    7h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
ACC     54h C=X        ACC     1Eh C=1
```

ПРИМЕР 2

```
MPYS    *, AR5 ; (PM = 0)
До выполнения          После выполнения
ARP     4              ARP     5
AR3     30Dh           AR3     30Dh
Память данных          Память данных
30Dh    7h             30Dh    7h
TREG0   6h             TREG0   6h
P       36h            P       2Ah
ACC     54h C=X        ACC     1Eh C=1
```

MPYU – беззнаковое умножение

СИНТАКСИС Прямая MPYU dma

Косвенная MPYU {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	1	See Section 5.2						

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ОПЕРАНДЫ $0 \le$ dma ≤ 127

$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC

беззнаковый(TREG0) * беззнаковый(dma) -> P

Не зависит от SXM.

ОПИСАНИЕ Беззнаковое содержимое TREG0 умножается на беззнаковое содержимое ячейки памяти данных. Результат пишется в регистр P. Умножитель обрабатывает операнды как знаковые 17-разрядные со старшим битом 0. Сдвигатель при чтении регистра P всегда выполняет расширение знака когда PM = 3 (сдвиг на 6 битов вправо). Поэтому такой режим сдвига не может использоваться, если требуется беззнаковый результат. MPYU можно использовать для операций с повышенной точностью.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p

External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Operand	Cycles for a Repeat (RPT) Execution			
	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
MPYU    DAT16 ; (DP = 4)
До выполнения           После выполнения
Память данных           Память данных
210h    0FFFFh          210h    0FFFFh
TREG0   0FFFFh          TREG0   0FFFFh
P       1h              P       0FFFE0001h
```

ПРИМЕР 2

```
MPYU    *, AR2

До выполнения           После выполнения
ARP     5               ARP     6
AR2     210h            AR2     210h
Память данных           Память данных
210h    0FFFFh          210h    0FFFFh
TREG0   0FFFFh          TREG0   0FFFFh
P       1h              P       0FFFE0001h
```

NEG – изменить знак аккумулятора

СИНТАКСИС NEG

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
ACC * -1 -> ACC
```

Зависит от OVM; влияет на OV и C.

ОПИСАНИЕ Содержимое аккумулятора заменяется его дополнением до 2. При выполнении NEG к 80000000h устанавливается бит OV. Если OVM = 1, содержимое аккумулятора заменяется 7FFFFFFFh. Если OVM = 0, результат 80000000h. Бит C устанавливается в 0 при ненулевом результате и в 1 при нулевом.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
NEG     ;( OVM = X )
До выполнения              После выполнения
ACC 0FFFFF288h C=X,OV=X    ACC 0DD8h C=OV=0
```

ПРИМЕР 2
```
NEG     ;( OVM = 0 )
До выполнения              После выполнения
ACC 80000000h C=X,OV=X     ACC 80000000h C=0,OV=1
```

ПРИМЕР 3
```
NEG     ;( OVM = 1 )
До выполнения              После выполнения
ACC 80000000h C=X,OV=X     ACC 7FFFFFFFh C=0,OV=1
```

NMI - немаскируемое прерывание

СИНТАКСИС NMI

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	0	0	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
```
(PC) + 1 -> стек
24 -> PC
```

ОПИСАНИЕ Программный счетчик устанавливается на вектор немаскируемого прерывания 24h. Эффект от инструкции тот же что и от аппаратного немаскируемого прерывания.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

ПРИМЕР
```
NMI  ; Управление передается по адресу 24h и PC+1
     ; помещается в стек.
```

NOP – нет операции

СИНТАКСИС NOP

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ОПИСАНИЕ Никаких операций не выполняется. NOP влияет только на PC. NOP – это то же самое, что и MAR с прямой адресацией и dma = 0h. NOP используется как заполнитель или временная инструкция при создании программы.

NORM – нормализация содержимого аккумулятора

СИНТАКСИС NORM {ind}

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
if(ACC == 0)
   TC -> 1;
else if( (ACC(31) xor ACC(30)) == 0){
        TC -> 0,
        (ACC) * 2 -> ACC;
        Заданная модификация текущего AR;
     }
     else
        TC -> 1.
```

Зависит от TC; влияет на TC.

ОПИСАНИЕ NORM предназначена для нормализации знакового содержимого аккумулятора. Нормализация числа с фиксированной точкой разделяет его на мантиссу и экспоненту. Для этого должен быть найден размер числа с расширенным знаком. Выполняется XOR битов 31 и 30 ACC, чтобы определить, является ли бит 30 частью числа или расширения знака.

Если они одинаковы, оба бита указывают на знаковое Расширение, поэтому аккумулятор сдвигается влево для удаления лишнего знакового бита.
Текущий AR модифицируется заданным образом для генерации размера экспоненты. Предполагается что текущий AR инициализируется перед началом нормализации. Модификация текущего AR по умолчанию – инкремент.
Для законченной нормализации 32-битного числа может потребоваться многократное выполнение NORM. Хотя использование NORM с RPT не обеспечивает выход из цикла при окончании нормализации, никаких операций не выполняется в остатке цикла. NORM работает с положительными и отрицательными значениями.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
NORM    *+
До выполнения           После выполнения
ARP     2               ARP     2
AR2     0h              AR2     01h
ACC 0FFFFF001h TC=X     ACC 0FFFE002h TC=0
```

ПРИМЕР 2

```
31-битовая нормализация:
MAR     *, APR1  ; AR1 используется для хранения экспоненты
LAR     AR1, #0h ; очистка счетчика экспоненты
LOOP    NORM    *+       ; нормализация одного бита
        BCND    LOOP, NTC ; Если TC = 0, размер еще не найден
```

ПРИМЕР 3

```
15-битовая нормализация
MAR     *, APR1  ; AR1 используется для хранения экспоненты
LAR     AR1, #0fh ; инициализация счетчика экспоненты
RPT     #14      ; Задана 15-битная нормализация (4-битная
                 ; экспонента и 16-битовая мантисса
NORM    *-       ; NORM автоматически заканчивает
                 ; сдвиг, когда найден первый значимый
                 ; бит, выполняя NOP до выхода из цикла
```

Метод из примера 2 используется для нормализации 32-битового числа и задает 5-битовую экспоненту.
Третий метод используется для нормализации 16-битового числа и задает 4-битовую экспоненту.
Если число требует малой степени нормализации, второй пример может быть предпочтительнее третьего.
В примере 2 цикл выполняется до завершения нормализации. Пример 3 всегда выполняет 15 циклов. Пример 2 более эффективен, если требуемое количество сдвигов 5 или меньше. Если требуемое количество сдвигов 6 или больше, пример 3 эффективнее.

Результирующее значение в дополнительном регистре не будет действительной экспонентой числа во всех случаях, однако может быть использовано для нахождения экспоненты.

OPL – OR значения памяти данных с DMBR или длинным непосредственным

СИНТАКСИС

Прямая	OPL [#lk,] dma [,сдвиг1]
Косвенная	OPL [#lk,] {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	1	0	dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
lk – 16-битовая константа

ВЫПОЛНЕНИЕ

lk не задана:
(PC) + 1 -> PC
(dma) OR (DMBR) -> dma

lk задана:
(PC) + 1 -> PC
(dma) OR lk -> dma

ОПИСАНИЕ

Если длинная непосредственная константа задана, выполняется OR со значением памяти данных. Если длинная непосредственная константа не задана, второй операнд берется из DMBR. Результат всегда сохраняется по тому же адресу (dma) памяти данных. ACC не меняется.

СЛОВА

1 (длинная непосредственная константа не задана)
2 (длинная непосредственная константа задана)

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	2n-2	2n-2	2n–2, 2n+1†	2n-2+p
External	4n-2+2nd	4n-2+2nd	4n-2+2nd	4n+1+2nd+p

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n-1	2n-1	2n–1, 2n+2†	2n-1+2p
External	4n-1+2nd	4n-1+2nd	4n-1+2nd	4n+2+2nd+2p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
OPL     DAT10   ;(DP = 6)
До выполнения           После выполнения
DMBR    0FFF0h          DMBR    0FFF0h
Память данных           Память данных
30Ah    0001h           30Ah    0FFF1h
```

ПРИМЕР 2
```
OPL     #0FFFh, DAT10   ;(DP = 6)
До выполнения           После выполнения
Память данных           Память данных
30Ah    0001h           30Ah    0FFF1h
```

ПРИМЕР 3
```
OPL     *, AR6
До выполнения           После выполнения
ARP     3               ARP     6
AR3     300h            AR3     300h
DMBR    0F0h            DMBR    0F0h
Память данных           Память данных
300h    0Fh             300h    0FFh
```

ПРИМЕР 4
```
OPL     #1111h, *, AR3
До выполнения           После выполнения
ARP     6               ARP     3
AR3     306h            AR3     306h
Память данных           Память данных
305h    0Eh             306h    111Fh
```

OR – OR с аккумулятором

СИНТАКСИС

Прямая	OR dma
Косвенная	OR {ind} [,новый ARP]
Длинная непосредственная	OR #lk [,сдвиг]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	1	See Section 5.2						

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	0	0	SHFT †			
16-Bit Constant															

Long immediate addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	1	0
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ сдвиг ≤ 16
lk – 16-битовая константа

ВЫПОЛНЕНИЕ

Прямая или косвенная адресация:
(PC) + 1 -> PC
(ACC(15-0)) OR (dma) -> ACC(15-0)
(ACC(31-16)) -> ACC(31-16)

Непосредственная адресация:
(PC) + 2 -> PC
(ACC) OR lk<<сдвиг -> ACC

Не зависит от SXM

ОПИСАНИЕ

Выполняется OR аккумулятора со значением памяти данных или со сдвинутой влево длинной константой. Все биты операнда, не занятые данными, заполняются 0 до 32 битного значения независимо от SXM так, старшее слово аккумулятора не меняется при прямой или косвенной адресации или непосредственной адресации без сдвига. При сдвиге младшие биты операнда заполняются 0.

СЛОВА

1 (прямая или косвенная адресация)
2 (длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
OR     DAT8   ;(DP = 8)
До выполнения          После выполнения
Память данных          Память данных
408h    0F000h         408h    0F000h
ACC     100002h C=X    ACC     10F002H C=X
```

ПРИМЕР 2

```
OR     *, AR0
До выполнения          После выполнения
ARP     1              ARP     0
AR1     300h           AR1     300h
Память данных          Память данных
300h    01111h         300h    01111h
ACC     222h C=X       ACC     1333H C=X
```

ПРИМЕР 3

```
OR     #08111h
До выполнения          После выполнения
ACC     0FF0000h C=X   ACC     0FF1100h C=X
```

ORB – OR ACCB с ACC

СИНТАКСИС ORB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) OR (ACCB) -> ACC

ОПИСАНИЕ Выполняется OR ACC с ACCB. Результат пишется в ACC.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР ORB

До выполнения		После выполнения	
ACC	5555555h C=X	ACC	5555557h C=X
ACCB	0000002h	ACCB	0000002h

OUT – вывод данных в порт

СИНТАКСИС Прямая `OUT dma , pa`

Косвенная `OUT {ind}, pa [,новый ARP]`

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
$0 \le dma \le 127$
$0 \le новый\ ARP \le 7$
$0 \le pa \le 65535$

ВЫПОЛНЕНИЕ
(PC) + 2 -> PC
адрес порта -> адресная шина (A15–A0)
(dma) -> шина данных (D15–D0)

ОПИСАНИЕ OUT пишет 16-битовое значение из памяти данных в порт ввода/вывода. На линия ~IS устанавливается низкий уровень для индикации доступа к портам ввода/вывода; ~STRB, R/~W и READY работают так же, как при записи во внешнюю память. Обратите внимание, что адреса портов PA0-PA15 – отображаемые в память, а остальные – нет.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$	$5+io_{dst}+2p_{code}$
Source: SARAM	$3+io_{dst}$	$3+io_{dst}$	$3+io_{dst}$, $4+io_{dst}$ †	$5+io_{dst}+2p_{code}$
Source: External	$3+d_{src}+io_{dst}$	$3+d_{src}+io_{dst}$	$3+d_{src}+io_{dst}$	$6+d_{src}+io_{dst}+2p_{code}$

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+3+nio_{dst}+2p_{code}$
Source: SARAM	$3n+nio_{dst}$	$3n+nio_{dst}$	$3n+nio_{dst}$, $3n+1+nio_{dst}$†	$3n+3+nio_{dst}+2p_{code}$
Source: External	$5n-2+nd_{src}+nio_{dst}$	$5n-2+nd_{src}++nio_{dst}$	$5n-2+nd_{src}+nio_{dst}$	$5n+1+nd_{src}+nio_{dst}+2p_{code}$

† If the operand and the code are in the same SARAM block

ПРИМЕР 1
```
OUT     DAT0, PA7 ; (DP = 4) Вывод слова данных из памяти
                  ; данных по адресу 200h в порт 7
```

ПРИМЕР 2
```
OUT     *, PA15 ; Вывод слова данных, указанного текущим
                ; дополнительным регистром в порт 15
```

PAC – загрузить P в аккумулятор

СИНТАКСИС PAC

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(сдвинутый регистр P) -> ACC

Зависит от PM.

ОПИСАНИЕ Содержимое регистра P, сдвинутое как указано в битами P, загружается в аккумулятор.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР PAC

До выполнения		После выполнения	
P	144h	P	144h
ACC	23h C=X	ACC	144h C=X

POP – вытолкнуть вершину стека в аккумулятор

СИНТАКСИС POP

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	0	0	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(TOS) -> ACC(0-15)
0 -> ACC(31-16)
Стек проталкивается на 1 уровень.

ОПИСАНИЕ Содержимое вершины стека (TOS) копируется в младшее слово аккумулятора, затем стек проталкивается на 1 уровень. Старшее слово аккумулятора обнуляется.
Аппаратный стек – первый вошел, последний вышел на 8 позиций. При POP каждое значение стека копируется вверх на 1 позицию. После POP внизу стека будет 2 одинаковых значения. Поскольку каждое значение стека копируется, после более 7 POP (POP, POPD, RETE, RETI,RET) все 7 уровней стека будут заполнены одним значением. Для обнаружения переполнения стека ничего не предпринимается.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР
```
POP
До выполнения          После выполнения
ACC 82H C=X            ACC     45h C=X

Stack 45h              Stack 16h
      16h                    7h
      7h                     33h
      33h                    42h
      42h                    56h
      56h                    37h
      37h                    61h
      61h                    61h
```

POPD – вытолкнуть вершину стека в память данных

СИНТАКСИС Прямая POPD dma
Косвенная POPD {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(TOS) -> dma
Стек проталкивается на 1 уровень.

ОПИСАНИЕ Содержимое вершины стека (TOS) копируется в заданную ячейку памяти, затем стек проталкивается на 1 уровень. Для обнаружения переполнения стека ничего не предпринимается.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
POPD DAT10 ;(DP = 8)
До выполнения          После выполнения
Память данных          Память данных
40Ah    55h            40Ah    45h
Stack 45h              Stack 16h
      16h                    7h
      7h                     33h
      33h                    42h
      42h                    56h
      56h                    37h
      37h                    61h
      61h                    61h
```

ПРИМЕР 2 POPD *+, AR1

До выполнения		После выполнения	
ARP	0	ARP	1
AR0	300h	AR0	301h
Память данных		Память данных	
300h	55h	300h	45h
Stack	45h	Stack	16h
	16h		7h
	7h		33h
	33h		42h
	42h		56h
	56h		37h
	37h		61h
	61h		61h

PSHD - поместить память данных в вершину стека

СИНТАКСИС Прямая PSHD dma
Косвенная PSHD {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ (dma) -> TOS
(PC) + 1 -> PC
Все ячейки стека вниз на 1 уровень.

ОПИСАНИЕ Содержимое памяти данных помещается на вершину стека.
Стек проталкивается вниз (см. описание PUSH).
Первое значение стека теряется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

† If the operand and the code are in the same SARAM block

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

† If the operand and the code are in the same SARAM block

ПРИМЕР 1

```
PSHD DAT10 ;(DP = 8)
До выполнения          После выполнения
Память данных          Память данных
40Ah    55h            40Ah    55h
Stack 45h              Stack 55h
      16h                    45h
      7h                     16h
      33h                    7h
      42h                    33h
      56h                    42h
      37h                    56h
      61h                    37h
```

ПРИМЕР 2 POHD *, AR1

До выполнения		После выполнения	
ARP	0	ARP	1
AR0	300h	AR0	300h
Память данных		Память данных	
300h	55h	300h	55h
Stack	45h	Stack	55h
	16h		45h
	7h		16h
	33h		7h
	42h		33h
	56h		42h
	37h		56h
	61h		37h

PUSH – поместить в вершину стека младшее слово аккумулятора

СИНТАКСИС PUSH

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
Все ячейки стека вниз на 1 уровень.
ACC(15-0) -> TOS

ОПИСАНИЕ Содержимое младшего слова аккумулятора копируется в вершину аппаратного стека. Стек проталкивается вниз до копирования младшего слова аккумулятора. Аппаратный стек – первый вошел, последний вышел на 8 позиций. Если произведено более 8 PUSH, (CALA, CALL, CC, PSHD, PUSH) первое записанное в стек значение теряется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Repeat (RPT) Execution			
ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution			
ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 PUSH

До выполнения		После выполнения	
ACC	7h C=X	ACC	7h C=X
Stack	45h	Stack	7h
	16h		45h
	7h		16h
	33h		7h
	42h		33h
	56h		42h
	37h		56h
	61h		37h

RETC – условный возврат

СИНТАКСИС Прямая RETC[D] [усл.1][, усл. 2] […

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	TP†		ZLVC†				ZLVC†t			

RETCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	TP†		ZLVC†				ZLVC†			

ОЕРАНДЫ Условия:

ACC=0	EQ
ACC!=0	NEQ
ACC<0	LT
ACC<=0	LEQ
ACC>0	GT
ACC>=0	GEQ
C=0	NC
C=1	C
OV=0	NOV
OV=1	OV
~BIO	BIO
TC=0	NTC
TC=1	TC
Без условия	UNC

ВЫПОЛНЕНИЕ

```
if( условие (я) ){
   (TOS) -> PC;
   стек вниз на 1 уровень;
}
else
   продолжить
```

ОПИСАНИЕ Стандартный возврат RET происходит при выполнении условий. Должны выполняться все заданные условия. Для задержанной версии RETC – RETCD перед выполнением возврата выбираются и выполняются 2 1-словных или 1 2-словная инструкция.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Condition	ROM	DARAM	SARAM	External Memory
True	4	4	4	4+3p†
False	2	2	2	2+p

Cycles for a Single Instruction для RETCD

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

ПРИМЕР 1

```
RETC GEQ, NOV ; Выполнить возврат при ACC ≤ 0 и
              ; OV = 0
```

ПРИМЕР 2

```
RETCD   C      ;  Выполнить возврат при установленном
MAR     *, 4   ;  бите переноса. Перед возвратом выполняются
LAR     AR3, #1h ;2 команды
```

RET – возврат из подпрограммы

СИНТАКСИС Прямая RET[D]

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0

RETD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (TOS) -> PC;
стек вверх на 1 уровень;

ОПИСАНИЕ Содержимое вершины стека копируется в PC. Стек вверх на 1 уровень. RET используется с CALA, CALL и CC для подпрограмм.
Для задержанной версии RET – RETD перед выполнением возврата выбираются и выполняются 2 1-словных или 1 2-словная инструкция.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

Cycles for a Single Instruction для RETD

ROM	DARAM	SARAM	External Memory
2	2	2	2+p

ПРИМЕР 1

```
RET
До выполнения          После выполнения
PC      96h            PC      45h
Stack 45h              Stack 16h
      16h                    7h
      7h                     33h
      33h                    42h
      42h                    56h
      56h                    37h
      37h                    61h
      61h                    61h
```

ПРИМЕР 2

```
RETD
MAR     *,4
LACC    #1h
До выполнения          После выполнения
PC      96h            PC      45h
ARP     0              ARP     4
ACC     0              ACC     1
Stack 45h              Stack 16h
      16h                    7h
      7h                     33h
      33h                    42h
      42h                    56h
      56h                    37h
      37h                    61h
      61h                    61h
```

RETE – разрешение прерываний и возврат из прерывания

СИНТАКСИС Прямая RETE

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	0	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (TOS) -> PC;
стек вверх на 1 уровень;
1 -> INTM ; глобальное разрешение прерываний

ОПИСАНИЕ Содержимое вершины стека копируется в PC. Стек вверх на 1 уровень. RETE автоматически устанавливает бит глобального разрешения прерываний (INTM в ST0) и восстанавливает теневые значения регистров (мс. RETI). RETE – эквивалент установки INTM и возврата RETI.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

ПРИМЕР 1 RETE

До выполнения		После выполнения	
PC	96h	PC	45h
ST0	xx4h	ST0	xx6xh
Stack	45h	Stack	16h
	16h		7h
	7h		33h
	33h		42h
	42h		56h
	56h		37h
	37h		61h
	61h		61h

RETI - возврат из прерывания

СИНТАКСИС Прямая RETI

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (TOS) -> PC;
стек вверх на 1 уровень;

ОПИСАНИЕ Содержимое вершины стека копируется в PC. Стек вверх на 1 уровень. RETI автоматически восстанавливает теневые значения регистров. Восстанавливаются следующие регистры: ACC, ACCB, PREG, ST0, ST1, PMST, ARCR,INDX, TREG0, TREG1, TREG2.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

ПРИМЕР 1

```
RETI
До выполнения          После выполнения
PC      96h            PC      45h
Stack 45h              Stack 16h
      16h                    7h
      7h                     33h
      33h                    42h
      42h                    56h
      56h                    37h
      37h                    61h
      61h                    61h
```

ROL – циклический сдвиг аккумулятора влево

СИНТАКСИС Прямая ROL

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(ACC(31)) -> C
(ACC(30-0)) -> ACC(31-1)
(C до ROL) -> ACC(0)

Влияет на C, зависит от C.
Не зависит от SXM.

ОПИСАНИЕ ROL осуществляет циклический сдвиг аккумулятора влево. Старший бит аккумулятора сдвигается в бит переноса, старое значение C помещается в младший бит аккумулятора.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 ROL

До выполнения После выполнения
ACC 0B0001234h C=0 ACC 60002468h C=1

ROLB – циклический сдвиг ACCB и аккумулятора влево

СИНТАКСИС Прямая ROLB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
```
(PC) + 1 -> PC
C -> ACCB(0)
(ACCB(30-0)) -> ACCB(31-1)
(ACCB(31)) -> ACC(0)
(ACC(30-0)) -> ACC(31-1)
(ACC(31)) -> C
```

Влияет на C, зависит от C.

ОПИСАНИЕ ROLB осуществляет 65-битный циклический сдвиг. Содержимое ACC и ACCB сдвигается влево на 1. Старший бит аккумулятора сдвигается в бит переноса, старое значение C помещается в младший бит ACCB.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
ROLB
До выполнения           После выполнения
ACC     08080808h C=0   ACC     10101011h C=1
ACCB    0FFFFFFFEh      ACCB    0FFFFFFFDh
```

ROR – циклический сдвиг аккумулятора вправо

СИНТАКСИС Прямая ROL

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	1	0	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(ACC(0)) -> C
(ACC(31-1)) -> ACC(30-0)
(C до ROL) -> ACC(31)

Влияет на C, зависит от C.
Не зависит от SXM.

ОПИСАНИЕ ROK осуществляет циклический сдвиг аккумулятора вправо. Младший бит аккумулятора сдвигается в бит переноса, старое значение C помещается в старший бит аккумулятора.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 ROL

До выполнения		После выполнения	
ACC	0B0001234h C=0	ACC	5800091Ah C=1

RORB - циклический сдвиг ACCB и аккумулятора вправо

СИНТАКСИС [метка] RORB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
```
(PC) + 1 -> PC
C -> ACC(31)
(ACC(31-1)) -> ACC(30-0)
(ACCB(0)) -> ACC(31)
(ACCB(31-1)) -> ACCB(30-0)
(ACCB(0)) -> C
```

Влияет на C, зависит от C.

ОПИСАНИЕ RORB осуществляет 65-битный циклический сдвиг. Содержимое ACC и ACCB сдвигается вправо на 1. Младший бит ACCB сдвигается в бит переноса, старое значение C помещается в старший бит аккумулятора.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
RORB
До выполнения            После выполнения
ACC     08080808h C=0    ACC     084040404h C=1
ACCB    0FFFFFFFEh       ACCB    7FFFFFFFh
```

RPT – повторить следующую инструкцию

СИНТАКСИС

Прямая	RPT dma
Косвенная	RPT {ind} [,новый ARP]
Короткая непосредственная	RPT #k
Длинная непосредственная	RPT #дk

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	8-Bit Constant							

Long immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	0	0
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ k ≤ 255
0 ≤ lk ≤ 65535

ВЫПОЛНЕНИЕ

Прямая или косвенная адресация:
(PC) + 1 -> PC
(dma(7-0)) -> RPTC

Короткая непосредственная адресация:
(PC) + 1 -> PC
k -> RPTC

Длинная непосредственная адресация:
(PC) + 2 -> PC
lk -> RPTC

ОПИСАНИЕ

Счетчик повтора (RPTC) загружается в соответствии с используемым способом адресации. Команда, следующая за RPT, повторяется N=(RPTC)+1 раз. Т.к. RPTC не сохраняется при переключении контекста, циклы повтора, как и многоцикловые инструкции, непрерываемы. При сбросе RPTC устанавливается в 0.
RPT полезна для операция перемещения блоков, умножения-аккумулирования, нормализации и т. д.

СЛОВА

1 (Прямая, косвенная или короткая непосредственная адресация)
2 (Длинная непосредственная адресация)

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Single Instruction (immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1

```
RPT     DAT137 ; (DP = 31)
До выполнения           После выполнения
Память данных           Память данных
0FFFh   0Ch             0FFFh   0Ch
RPTC    0               RPTC    0Ch
```

ПРИМЕР 2

```
RPT     *, AR1
До выполнения           После выполнения
ARP     0               ARP     1
AR0     300h            AR0     300h
Память данных           Память данных
300h   0FFFh            300h    0FFFh
RPTC    0               RPTC    0FFFh
```

ПРИМЕР 3

```
RPT     #1 ; повторить 2 раза
До выполнения           После выполнения
RPTC    0               RPTC    1h
```

ПРИМЕР 4

```
RPT     #1111h ; повторить 4370 раз
До выполнения           После выполнения
RPTC    0               RPTC    1111h
```

RPTB – повтор блока

СИНТАКСИС RPTB pma

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	1	0
16-Bit Constant															

ОПЕРАНДЫ $0 \leq pma \leq 65535$

ВЫПОЛНЕНИЕ
```
1 -> BRAF
PC+2 -> PASR
pma -> PAER
```

ОПИСАНИЕ RPTB позволяет выполнить блок команд заданное в BRCR количество раз без затрат на управление циклом. До выполнения RPTB должен быть загружен BRCR. При выполнении RPTB в указатели начала и конца блока PASR и PAER загружаются соответственно PC+2 и pma. Бит активности повтора блока BRAF устанавливается в 1. Повтор блока может быть прекращен очисткой BRAF. Количество итераций цикла N=(BRCR)+1.
RPTB прерываема, однако не может быть продолжена, если BRCR, PAER и PASR не сохранены и не восстановлены соответствующим образом. В блоки повтора RPTB могут включаться повторы одной инструкции (RPT, RPTZ).

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР
```
        SPLK    iterations_minus_1, BRCR
        RPTB    end_block – 1
        LACC    DAT1
        ADD     DAT2
        SACL    DAT1
end_block
```

RPTZ – очистить PREG и ACC и повторить следующую инструкцию

СИНТАКСИС Длинная непосредственная RPTZ #lk

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	1	0	0	0	1	1	0
16-Bit Constant															

ОПЕРАНДЫ $0 \leq lk \leq 65535$

ВЫПОЛНЕНИЕ
```
0 -> ACC
0 -> PREG
(PC) + 1 -> PC
lk -> RPTC
```

ОПИСАНИЕ RPTZ очищает ACC и PREG и повторяет следующую инструкцию N=lk+1 раз. RPTZ – эквивалент следующей последовательности инструкций :

```
MPY     #0
PAC
RPT     #<lk>
```

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР
```
PRTZ    #7FFh  ; обнуляем P и ACC
MACD    pma, *+ ; повтор MACD 2048 раз
```

SACB – сохранить ACC в ACCB

СИНТАКСИС SACB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) -> ACCB

ОПИСАНИЕ Содержимое ACC копируется в ACCB.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 SACB

До выполнения		После выполнения	
ACC	7C638421h	ACC	7C638421h
ACCB	5h	ACC	7C638421h

SACH – сохранить старшее слово ACC со сдвигом в память данных

СИНТАКСИС
Прямая SACH dma [,сдвиг]
Косвенная SACH {ind} [,сдвиг[,новый ARP]]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	SHF †			0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	SHF †			1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ сдвиг ≤ 7 (по умолчанию 0)

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(ACC) << сдвиг -> dma

Не зависит от SXM

ОПИСАНИЕ
SACH копирует весь аккумулятор в сдвигатель, где он сдвигается влево на 0-7 бит, а затем старшие 16 бит копируются в память. Значение аккумулятора не меняется.

СЛОВА
1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

ПРИМЕР 1

```
SACH    DAT10, 1 ; (DP = 4)
До выполнения           После выполнения
ACC     4208001h C=X    ACC     4208001h C=X
Память данных           Память данных
20Ah    00h             20Ah    0841h
```

ПРИМЕР 2

```
SACH    *+, 0, AR2
До выполнения           После выполнения
ARP     1               ARP     2
AR2     0300h           AR2     0301h
ACC     4208001h C=X    ACC     4208001h C=X
Память данных           Память данных
300h    0h              300h    0420h
```

SACL – загрузить младшее слово ACC со сдвигом в память данных

СИНТАКСИС Прямая SACL dma [,сдвиг]
Косвенная SACL {ind} [,сдвиг[,новый ARP]]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	SHF †			0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	SHF †			1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ сдвиг ≤ 7 (по умолчанию 0)

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
((ACC) << сдвиг)(15-0) -> dma

Не зависит от SXM

ОПИСАНИЕ Младшее слово аккумулятора сдвигается влево на 0-7 бит, а затем копируется в память. Младшие биты заполняются 0. Значение аккумулятора не меняется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

ПРИМЕР 1
```
SACL    DAT10, 1 ; (DP = 4)
До выполнения           После выполнения
ACC     7C638421 C=X    ACC     7C638421 C=X
Память данных           Память данных
20Ah    00h             20Ah    0841h
```

ПРИМЕР 2
```
SACH    *, 0, AR7
До выполнения           После выполнения
ARP     6               ARP     7
AR2     0300h           AR2     0300h
ACC     00FF8421 C=X    ACC     00FF8421 C=X
Память данных           Память данных
300h    5h              300h    8421h
```

SAMM – Сохранить аккумулятор в отображаемый в памяти регистр

СИНТАКСИС Прямая SAMM dma
Косвенная SAMM {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ 0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
ACC -> dma(0-7)

ОПИСАНИЕ Младшее слово аккумулятора копируется в отображаемый в память регистр. 9 старших битов адреса данных обнуляются независимо от текущего значения DP или старших 9 битов текущего AR. Эта инструкция позволяет записать аккумулятор в любую ячейку памяти данных в странице 0 без модификации поля DP в регистре статуса ST0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
MMR†	1	1	1	1+p
MMPORT	$2+io_{dst}$	$2+io_{dst}$	$2+io_{dst}$	$4+io_{dst}$

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
MMR‡	n	n	n	n+p
MMPORT	$2+nio_{dst}$	$2+nio_{dst}$	$2+nio_{dst}$	$2n+2+p+p$ nio_{dst}

ПРИМЕР 1
```
SAMM    PRD ; (DP = 6)
До выполнения          После выполнения
ACC     22221376h      ACC     5555h
PRD       5h           PRD       80h
Память данных          Память данных
325h      0Fh          325h      0Fh
```

ПРИМЕР 2
```
SAMM   *, AR2  ;(BMAR = 1Fh)
До выполнения          После выполнения
ARP     7              ARP     2
AR0     31Fh           AR0    31Fh
ACC      80h           ACC     80h
BMAR     0h            BMAR    80h
Память данных          Память данных
31Fh      11h          31Fh      11h
значение ячейки  25h.
```

SAR - сохранить дополнительный регистр

СИНТАКСИС
Прямая SAR ARx, dma
Косвенная SAR ARx, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	ARX †			0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	ARX †			1	See Section 5.2						

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ ar ≤ 7
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(ARx) -> (dma)

ОПИСАНИЕ
Содержимое заданного дополнительного регистра пишется по заданному адресу.
При модификации содержимого текущего дополнительного регистра в режиме косвенной адресации SAR ARx(где x = ARP) записывает значение дополнительного регистра до его изменения.
LAR и SAR могут быть использованы для загрузки и сохранения дополнительных регистров во время вызова подпрограмм или прерываний. Если дополнительный регистр не будет использоваться для косвенной адресации, LAR и SAR позволяют использовать его как дополнительный регистр хранения данных, особенно для обмена значениями ячеек памяти без изменения содержимого ACC.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

ПРИМЕР 1

```
SAR    AR0, DAT30 ; (DP = 6)
До выполнения          После выполнения
Память данных          Память данных
31Eh    18h            31Eh    37h
AR0     37h            AR0     37h
```

ПРИМЕР 2 SAR AR4, *-

До выполнения		После выполнения	
Память данных		Память данных	
401h	0h	401h	401h
AR4	401h	AR4	402h

SATH – Циклический сдвиг ACC как задано в TREG1

СИТАКСИС SATH

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	1	0

ОПЕРАНДЫ нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
16 * (TREG1(4)) -> счетчик
(ACC) >> счетчик -> ACC

Зависит от SXM.

ОПИСАНИЕ Команда SATH выполняет 16 битовый циклический сдвиг аккумулятора вправо на величину, заданную 4 младшими битами TREG1.Если SXM = 0, старшие биты заполняются 0, в противном случае в них копируется ACC(31). SATH вместе с SATL обеспечивает 2-цикловый 0-31 битовый сдвиг вправо. Бит переноса не меняется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
SATH       ; (SXM = 0)
До выполнения                 После выполнения
ACC     FFFF0000h             ACC     0000FFFFh
TREG1   xx1xh                 TREG1   xx1xh
```

ПРИМЕР 2

```
SATH       ; (SXM = 1)
До выполнения                 После выполнения
ACC     FFFF0000h             ACC     0FFFFFFFFh
TREG1   xx1xh                 TREG1   xx1xh
```

SATL – Барабанный сдвиг ACC как задано в TREG1

СИТАКСИС SATL

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	U
1	0	1	1	1	1	1	0	0	1	0	1	1	0	1	1

ОПЕРАНДЫ нет

ИСПОЛНЕНИЕ
(PC) + 1 -> PC
(TREG1(3-0)) -> счетчик
(ACC) >> счетчик -> ACC

Зависит от SXM.

ОПИСАНИЕ Команда SATL выполняет циклический сдвиг аккумулятора вправо, если 4-й бит TREG1 = 1. Если 4-й бит TREG1 = 0, значение аккумулятора не меняется. Если SXM = 0, старшие биты заполняются 0, в противном случае в них копируется ACC(31). SATH вместе с SATL обеспечивает 2-цикловый 0-31 битовый правый сдвиг. Бит переноса не меняется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1
```
SATH       ; (SXM = 0)
До выполнения                После выполнения
ACC     FFFF0000h            ACC     3FFFC000h
TREG1     x2h                TREG1     x2h
```

ПРИМЕР 2
```
SATH       ; (SXM = 1)
До выполнения                После выполнения
ACC     FFFF0000h            ACC     0FFFFC000h
TREG1     x2h                TREG1     x2h
```

SBB – вычесть ACCB из ACC

СИНТАКСИС SBB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) – (ACCB) -> ACC

ОПИСАНИЕ Содержимое ACCB вычитается из ACC, результат пишется в ACC. ACCB не меняется. Если в результате вычитания генерируется заем, бит переноса устанавливается в 0; иначе бит C устанавливается в 1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 SBB

До выполнения		После выполнения	
ACC	20000000h C=X	ACC	10000000h C=1
ACCB	10000000h	ACC	10000000h

SBBB – вычесть ACCB и C из ACC

СИНТАКСИС SBBB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	U
1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) – (ACCB) – (~C) -> ACC

ОПИСАНИЕ Содержимое ACCB и логическая инверсия бита переноса вычитается из ACC, результат пишется в ACC. ACCB не меняется. Если в результате вычитания генерируется заем, бит переноса устанавливается в 0;иначе бит C устанавливается в 1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
SBBB
До выполнения            После выполнения
ACC     20000000h C=1    ACC     10000000h C=1
ACCB    10000000h        ACC     10000000h
```

ПРИМЕР 2

```
SBBB
До выполнения            После выполнения
ACC        98012h C=0    ACC           1h C=1
ACCB       98010h        ACC       98010h
```

SBRK – вычесть короткое непосредственное из дополнительного регистра

СИНТАКСИС SBRK #k

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	8-Bit Constant							

ОПЕРАНДЫ $0 \leq k \leq 255$

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
Текущий AR – 8-битовая положительная константа -> текущий AR

ОПИСАНИЕ 8-битовое непосредственное значение вычитается из текущего дополнительного регистра; результат заменяет старое содержимое дополнительного регистра. Вычитание выполняется в ARAU, непосредственное значение считается 8-битовым беззнаковым целым.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР SBRK 0FFh

До выполнения		После выполнения	
ARP	7	ARP	7
AR7	0h	AR7	0FF01h

SETC – установить управляющий бит

СИНТАКСИС SETC control_bit

КОД КОМАНДЫ

SETC OVM (Set overflow mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	1	1

SETC: SXM (Set sign extension mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	1	1

SETC: HM (Set hold mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	0	1

SETC TC (Set test/control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	0	1	1

SETC C (Set carry)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	1

SETC XF (Set external flag pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	1	1	0	1

SETC CNF (Set configuration control)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	1	0	1

SETC: INTM (Set interrupt mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	0	0	0	0	1

ОПЕРАНДЫ control_bit : бит ST0 или ST1(из следующего набора) :
(C, CNF, HM, IMTM, OVM, SXM, TC, XF)

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
1 -> control_bit

ОПИСАНИЕ Заданный управляющий бит устанавливается в 1.
Чтобы загрузить ST0 и ST1 может также быть
использована команда LST.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР SETC TC ; TC – бит 11 ST1

До выполнения		После выполнения	
ST1	x1xxh	st1	x9xxh

SFL – сдвиг ACC влево

СИНТАКСИС SFL

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	0	0	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
```
(PC) + 1 -> PC
(ACC(31)) -> C
(ACC(30-0)) -> ACC(31-1)
0 -> ACC(0)
```

Влияет на C.
Не зависит от SXM.

ОПИСАНИЕ SFL сдвигает весь ACC влево на 1 бит. Младший бит заполняется 0, а старший сдвигается в бит C.
В отличие от SFR SFL не зависит от SXM.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР SFL

```
До выполнения           После выполнения
ACC     0B0001234h C=X  ACC     60002468h C=1
```

SFLB – сдвиг ACCB и аккумулятора влево

СИНТАКСИС SFLB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ
```
(PC) + 1 -> PC
0 -> ACCB(0)
(ACCB(30-0)) -> ACCB(31-1)
(ACCB(31)) -> ACC(0)
(ACC(30-0)) -> ACC(31-1)
(ACC(31)) -> C
```

Влияет на C.
Не зависит от SXM.

ОПИСАНИЕ SFLB осуществляет 65-битный сдвиг комбинации ACC и ACCB влево на 1.
Старший бит аккумулятора сдвигается в бит переноса.
Младший бит ACCB заполняется 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1 SFLB

```
До выполнения            После выполнения
ACC     0B0001234h C=X   ACC     60002469h C=1
ACCB    0B0001234h       ACC     60002468h
```

SFR – сдвиг ACC вправо

СИНТАКСИС SFR

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	1	0	1	0

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
if(SXM == 0)
    0 -> ACC(31);
(ACC(31-1)) -> ACC(30-0)
ACC(0) -> C
```

Влияет на C.
Зависит от SXM.

ОПИСАНИЕ SFR сдвигает ACC вправо на 1 бит.
Если SXM = 1, выполняется арифметический сдвиг.
Знаковый (старший) бит не меняется и копируется в 30-й бит.
Если SXM = 0, выполняется логический сдвиг вправо.
Старшие биты заполняется 0.

Младший бит сдвигается в бит C.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
SFR  ; (SXM = 0)
До выполнения           После выполнения
ACC     0B0001234h C=X  ACC     5800091Ah C=0
```

ПРИМЕР 2

```
SFR  ; (SXM = 1)
До выполнения           После выполнения
ACC     0B0001234h C=X  ACC     D5800091Ah C=0
```

SFRB – сдвиг ACCB и аккумулятора вправо

СИНТАКСИС SFRB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	0	1	1	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ

```
(PC) + 1 -> PC
if(SXM == 0)
    0 -> ACC(31);
(ACCB(31-1)) -> ACCB(30-0)
(ACCB(0)) -> ACC(31)
(ACC(31-1)) -> ACC(30-0)
(ACC(0)) -> C
```

Влияет на C.
Зависит от SXM.

ОПИСАНИЕ SFLB осуществляет 65-битный сдвиг комбинации ACC и ACCB вправо на 1. Младший бит ACCB сдвигается в C. Если SXM = 1, выполняется арифметический сдвиг вправо. Знаковый (старший) бит ACC не меняется и копируется в 30-й бит.
Если SXM = 0, выполняется логический сдвиг влево. Старшие биты ACC заполняется 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
SFRB ;(SXM = 0)
До выполнения            После выполнения
ACC     0B0001235h C=X   ACC     5800091Ah C=0
ACCB    0B0001234h       ACC     0D800091Ah
```

ПРИМЕР 2

```
SFRB ;(SXM = 1)
До выполнения            После выполнения
ACC     0B0001234h C=X   ACC     0D800091Ah C=0
ACCB    0B0001234h       ACC     5800091Ah
```

SMMR – сохранить отображаемый в памяти регистр

СИНТАКСИС Прямая SMMR dma, #lk
Косвенная SMMR {ind}, #lk [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ lk ≤ 65535

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(dma(7-0)) -> lk

ОПИСАНИЕ Отображаемый в памятb регистр, указываемый 7 младшими битами памяти данных, пишется по адресууказанному длинной непосредственной константой.9 старших битов адреса данных обнуляются независимо от текущего значения DP или старших 9 битов текущего AR. Эта инструкция позволяет прочитать любую ячейку памяти данных в странице 0 без модификации поля DP в регистре статуса ST0.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM Source: MMR‡	2	2	2	$2+2p_{code}$
Destination: SARAM Source: MMR‡	2	2	2, 3†	$2+2p_{code}$
Destination: External Source: MMR‡	$3+d_{dst}$	$3+d_{dst}$	$3+d_{dst}$	$5+d_{dst}+2p_{code}$
Destination: DARAM Source: MMPORT	$3+io_{src}$	$3+io_{src}$	$3+io_{src}$	$4+io_{src}+2p_{code}$
Destination: SARAM Source: MMPORT	$3+io_{src}$	$3+io_{src}$	$3+io_{src}$, $4+io_{src}$†	$3+io_{src}+2p_{code}$
Destination: External Source: MMPORT	$4+io_{src}+d_{dst}$	$4+io_{src}+d_{dst}$	$4+io_{src}+d_{dst}$	$6+io_{src}+d_{dst}+2p_{code}$

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Destination: DARAM Source: MMR§	2n	2n	2n	$2n+2p_{code}$

Destination: SARAM Source: MMR§	2n	2n	2n, 2n+2†	$2n+2p_{code}$
Destination: External Source: MMR§	$3n+nd_{dst}$	$3n+nd_{dst}$	$3n+nd_{dst}$	$3n+3+nd_{dst}+2p_{code}$
Destination: DARAM Source: MMPORT	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$	$2n+1+nio_{src}+2p_{code}$
Destination: SARAM Source: MMPORT	$2n+nio_{src}$	$2n+nio_{src}$	$2n+nio_{src}$, $2n+2+nio_{src}$†	$2n+1+nio_{src}+2p_{code}$
Destination: External Source: MMPORT	$5n-2+nd_{dst}+nio_{src}$	$5n-2+nd_{dst}+nio_{src}$c	$5n-2+nd_{dst}+nio_{src}$	$5n+1+nd_{dst}+nio_{src}+2p_{code}$

```
ПРИМЕР 1        SMMR    CBSR, #307h ;( DP = 6, CBSR = 1Eh)
                До выполнения           После выполнения
                Память данных           Память данных
                307h    1376h           307h    5555h
                CBSR    5555h           CBSR    5555h

ПРИМЕР 2        SMMR    *, #307h, AR6 ; (CBCR = 1eh)
                До выполнения           После выполнения
                ARP     6               ARP     6
                AR0     0F01Eh          AR20    0F01Eh
                Память данных           Память данных
                307h    1376h           307h    5555h
                CBSR    5555h           CBSR    5555h
```

SPAC – вычесть регистр P из ACC

СИНТАКСИС SPAC

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	1

ОПЕРАНДЫ нет

ВЫПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) – сдвинутый регистр P -> ACC

Зависит от OVM, PM; влияет на OV и C.
Не зависит от SXM.

ОПИСАНИЕ Содержимое регистра P, сдвинутое как определено битами PM, вычитается из ACC. SPAC не зависит от SXM, всегда происходит расширение знака P.

SPAC является подфункцией LTS, MPYS, SQRS.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p

ПРИМЕР 1

```
SPAC    ; (PM = 0)
До выполнения           После выполнения
P       10000000h       P       10000000h
ACC     70000000h C=X   ACC     60000000h C=1
```

SPH – сохранить старшее слово регистра P

СИНТАКСИС

Прямая	SPH dma
Косвенная	SPH {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC
(выход сдвигателя PR(31-16)) -> dma

ОПИСАНИЕ

Старшее слово регистра P, сдвинутое как определено битами PM, записывается в память данных. Ни регистр P, ни аккумулятор не меняются. В старшие биты происходит расширение знака при выбранном режиме сдвига 6. При левом сдвиге младшие биты берутся из младшего слова P.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

ПРИМЕР 1

```
SPH    DAT3 ; (DP = 4, PM = 0)
До выполнения          После выполнения
P       0FE079844h     P       0FE079844h
Память данных          Память данных
203h    4567h          203h    0FE07h
```

ПРИМЕР 2

```
SPH    *, AR7 ; (PM=2)
До выполнения          После выполнения
ARP     6              ARP     7
AR6     203h           AR6     203h
Память данных          Память данных
203h    4567h          203h    0E079h
P       0FE079844h     P       0FE079844h
```

SPL – сохранить младшее слово регистра P

СИНТАКСИС Прямая SPL dma
Косвенная SPL {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	0	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
(выход сдвигателя PR(15-0)) -> dma

ОПИСАНИЕ Младшее слово регистра P, сдвинутое как определено битами PM, записывается в память данных. Ни регистр P, ни аккумулятор не меняются. Старшие биты берутся из старшего слова P при выбранном режиме сдвига 6. При левом сдвиге младшие биты заполняются 0.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1 1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+nd+p

ПРИМЕР 1
```
SPL    DAT3 ; (DP = 4, PM = 0)
До выполнения          После выполнения
Память данных          Память данных
203h    4567h          203h   08440h
P       0FE079844h     P       0FE079844h
```

ПРИМЕР 2
```
SPL    *, AR7 ;(PM=2)
До выполнения          После выполнения
ARP     6              ARP     7
AR6     203h           AR6     203h
Память данных          Память данных
203h    4567h          203h   09844h
P       0FE079844h     P       0FE079844h
```

SPLK – Сохранить длинный непосредственный операнд

СИНТАКСИС Прямая SPLK #lk, dma
Косвенная SPLK #lk, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	0	0	dma						
16-Bit Constant															

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
lk : 16-битовая константа

ВЫПОЛНЕНИЕ
(PC) + 1 -> PC
lk -> dma

ОПИСАНИЕ SPLK позволяет записать 16-битный операнд в любую ячейку памяти. PLU поддерживает эту манипуляцию битами независимо от АЛУ, поэтому ACC не меняется.

СЛОВА 2

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2, 3†	2+2p
External	3+d	3+d	3+d	5+d+2p

ПРИМЕР 1
```
SPLK    #7FFFh, DAT3 ; (DP = 6)
До выполнения          После выполнения
Память данных          Память данных
303h    7h             303h    7FFFh
```

ПРИМЕР 1
```
SPLK    #1111h, *+, AR4
До выполнения          После выполнения
ARP     0              ARP     4
AR4     300h           AR4     301h
Память данных          Память данных
300h    7h             300h    1111h
```

SPM – установить режим сдвига регистра P

СИНТАКСИС SPM константа

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	0	0	0	PM †	

ОПЕРАНДЫ (PC) + 1 -> PC
константа -> PM

Влияет на PM.
Не зависит от SXM

ОПИСАНИЕ 2 младших бита инструкции копируются в поле PM регистра статуса ST1. PM управляет режимом сдвига регистра P. Сдвигатель может сдвигать регистр P на 1 или 4 бита влево, или на 6 битов вправо. Комбинации битов следующие:

PM	Значение
00	без сдвига
01	P << 1
10	P << 4
11	P >> 6 с расширением знака

Левый сдвиг позволяет выравнивать результат. Правый сдвиг на 6 введен для обеспечения до 128 умножений с накоплением без опасности переполнения. PM может быть загружен инструкцией LST #1.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

ПРИМЕР

```
SPM     3 ; P >> 6 при вводе в АЛУ
```

SQRA - возвести в квадрат и аккумулировать предыдущий результат

СИНТАКСИС Прямая SQRA dma
Косвенная SQRA {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le$ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + 1 -> PC
(ACC) + (сдвинутый регистр P) -> ACC
(dma) -> TREG0
(dma) * (dma) -> P

Зависит от OVM, PM; влияет на OV и C.

ОПИСАНИЕ Значение из памяти загружается в TREG0 , возводится в квадрат и записывается в регистр P.
Предыдущий результат P-регистра, сдвинутый, как определено битами PM, прибавляется к ACC.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SQRA    DAT13 ; (DP = 6, PM = 0)
До выполнения          После выполнения
Память данных          Память данных
30Dh    0Fh            30Dh    0Fh
TREG0   6h             TREG0   0Fh
P       12Ch           P       0E1h
ACC     1F4h C=X       ACC     320h C=0
```

ПРИМЕР 2

```
SQRA    *, AR4 ; (PM = 0)
До выполнения          После выполнения
ARP     3              ARP     4
AR3     30Dh           AR3     30Dh
Память данных          Память данных
30Dh    Fh             30Dh    Fh
TREG0   6h             TREG0   0Fh
P       12Ch           P       0E1h
ACC     1F4h C=X       ACC     320h C=0
```

SQRS – возвести в квадрат и вычесть предыдущий результат

СИНТАКСИС

Прямая	SQRS dma
Косвенная	SQRS {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + -> PC
(ACC) – (сдвинутый регистр P) -> ACC
(dma) -> TREG0
(dma) * (dma) -> P

Зависит от OVM, PM; влияет на OV и C.

ОПИСАНИЕ

Значение из памяти загружается в TREG0 , возводится в квадрат и записывается в регистр P.
Предыдущий результат P-регистра, сдвинутый, как определено битом статуса PM, вычитается из ACC.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SQRS    DAT13 ; (DP = 6, PM = 0)
До выполнения          После выполнения
Память данных          Память данных
30Dh    08h            30Dh    08h
TREG0   1124h          TREG0   08h
P       190h           P       40h
ACC     1450h C=X      ACC     12C0h C=1
```

ПРИМЕР 2

```
SQRA    *, AR4 ; (PM = 0)
До выполнения          После выполнения
ARP     3              ARP     4
AR3     30Dh           AR3     30Dh
Память данных          Память данных
30Dh    08h            30Dh    08h
TREG0   1124h          TREG0   08h
P       190h           P       40h
ACC     1450h C=X      ACC     12C0h C=1
```

SST - сохранить регистр статуса

СИНТАКСИС

Прямая	SST #n, dma
Косвенная	SST #n, {ind} [,новый ARP]

КОД КОМАНДЫ

Direct addressing for SST#0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	0	dma						

Indirect addressing for SST#0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	0	1	See Section 5.2						

Direct addressing for SST#1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0	dma						

Indirect addressing for SST#1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ n ≤ 1
0 ≤ новый ARP ≤ 7

ВЫПОЛНЕНИЕ

(PC) + -> PC
(регистр статуса STn) -> dma

ОПИСАНИЕ

Регистр статуса STn пишется в память данных. В прямом режиме адресации STn всегда пишется в страницу 0 независимо от значения DP. Берется заданное в команде смещение от начала этой страницы. Регистр DP не меняется. Это позволяет записать регистр DP в памяти данных при прерывании и т. д. без необходимости изменения DP. В косвенном режиме адресации адрес памяти данных берется из выбранного дополнительного регистра (см. LST). В косвенном режиме адресации может быть адресована любая страница.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	2+d	2+d	2+d	4+d+p

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	2n+nd	2n+nd	2n+nd	2n+2+p+nd

```
ПРИМЕР 1        SST    #0, DAT96 ; (DP = 6)
                До выполнения          После выполнения
                ST0     0A408h         ST0     0A408h
                Память данных          Память данных
                60h     0Ah            60h     0A408h

ПРИМЕР 2        SST    #1, *, AR7
                До выполнения          После выполнения
                ARP     0              ARP     7
                AR3     300h           AR3     300h
                ST1     2580h          ST1     2580h
                Память данных          Память данных
                300h     0h            300h    2580h
```

SUB – вычесть из аккумулятора значение памяти данных

СИНТАКСИС

Прямая: SUB dma[,сдвиг]
Косвенная: SUB {индекс}[,сдвиг][,новый ARP]]
Короткая непосредственная: SUB #k
Длинная непосредственная: SUB #k[,сдвиг]

КОД КОМАНДЫ

Direct addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	SHFT †				0	dma						

Indirect addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	SHFT †				1	See Section 5.2						

Direct addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1	0	dma						

Indirect addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1	1	See Section 5.2						

Short immediate addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	8-Bit Constant							

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	1	0	SHFT †			
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ новый ARP ≤ 7
0 ≤ k ≤ 255
-32768 ≤ lk ≤ 32767
0 ≤ сдвиг2 ≤ 15 (по умолчанию 0)

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:
(PC) + 1 -> PC
(ACC) – [(dma) << сдвиг] -> ACC
Зависит от SXM и OVM; влияет на C и OV.

Короткая непосредственная адресация:

(PC) + 1 -> PC
(ACC) – k -> ACC
Зависит от OVM; влияет на C и OV.

Длинная непосредственная адресация:
(PC) + 2 -> PC
(ACC) – lk << сдвиг -> ACC
Зависит от OVM; влияет на C и OV.

ОПИСАНИЕ

Содержание ячейки памяти данных или непосредсвенная константа сдвигается влево вычитается из аккумулятора. В течение сдвига младшие разряды битов заполняются нулями. В старших разрядах происходит расширение знака, если SXM = 1, или они заполняются нулями, если SXM = 0. Результат запоминается в аккумуляторе.

Когда используется короткая непосредственная адресация, из аккумулятора вычитается 8-битовая константа, сдвиг не может быть задан, вычитание не зависит от SXM и инструкция неповторяема.

Если вычитание генерирует заем, бит переноса устанавливается в 0, в противном случае в 1. Если задан 16-битовый сдвиг, инструкция может лишь установить в 1 бит переноса, если вычитание генерирует перенос; в противном случае C не меняется.

СЛОВА 1 (Прямая, косвенная или короткая непосредственная адресация).
1000 (Длинная непосредственная адресация).

ЦИКЛЫ

Cycles for a Single Instruction (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

Cycles for a Single Instruction (short immediate addressing)

ROM	DARAM	SARAM	External Memory
1	1	1	1+p

Cycles for a Single Instruction (long immediate addressing)

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p

ПРИМЕР 1
```
SUB DAT80,1  ;(DP = 8,SXM = 0)
До выполнения                  После выполнения
Память данных                  Память данных
450h   11h                     450h   11h
ACC    24h  C=X                ACC     13h  C=1
```

ПРИМЕР 2
```
SUB *-,1,AR0  ;(SXM = 0)
До выполнения                  После выполнения
ARP     7                      ARP     0
AR7     0301h                  AR7     0300h
Память данных                  Память данных
301h    4h                     301h   42h
ACC     9h  C=X                ACC     01h  C=1
```

ПРИМЕР 3
```
SUB #8h   ; (SXM = 1)
До выполнения                  После выполнения
ACC     7h  C=X                ACC     0FFFFFFFh  C=0
```

```
ПРИМЕР 4        ADD #FFFFh,4   ; (SXM = 1)
                До выполнения                 После выполнения
                ACC     0FFFFh  C=X           ACC     0Fh  C=1
```

SUBB – вычесть из аккумулятора значение памяти данных и бит C

СИНТАКСИС

Прямая: SUBB dma

Косвенная: SUBB {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$

$0 \le n \le 7$

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:

(PC) + 1 -> PC

(ACC) – (dma) – (логическая инверсия C) -> ACC

Зависит от OVM; Не зависит от SXM; Влияет на C и OV.

ОПИСАНИЕ

Содержание ячейки памяти данных (dma) и логически инвертированный бит C вычитается из содержимого аккумулятора с подавлением знаковым расширением. Результат сохраняется в аккумуляторе. Бит C очищается, если результат вычитания вызывает заём, иначе бит C устанавливается в 1.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SUBB DAT5  ;(DP = 8)
До выполнения                  После выполнения
Память данных                  Память данных
405h   06h                     405h  06h
ACC    06h  C=0                ACC   FFFF FFFFh  C=0
```

ПРИМЕР 2

```
SUBB *
До выполнения                  После выполнения
ARP     6                      ARP     6
AR6     301h                   AR7     301h
301h    02h                    301h    02h
ACC     04h  C=1               ACC     02h  C=1
```

SUBC – Условное вычитание из аккумулятора значения памяти данных

СИНТАКСИС Прямая: SUBC dma
Косвенная: SUBC {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	See Section 5.2						

ОПЕРАНДЫ $0 \le dma \le 127$
$0 \le n \le 7$

ИСПОЛНЕНИЕ Прямая и косвенная адресация:

(PC) + 1 -> PC
(ACC) – ((dma)**x** 2^{15}) -> ALU выход

Если выход ALU ≥ 0: (выход ALU)**x** 2 + 1 -> ACC
Иначе (ACC)**x** 2 -> ACC

Не зависит от OVM и SXM; Влияет на C и OV.

ОПИСАНИЕ Инструкция SUBC выполняет вычитание по условию, которое может быть использовано для деления. Делитель в памяти данных. После завершения последней инструкции SUBC частное от деления в ACCL и остаток в ACCH.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SUBC DAT2  ;(DP = 6)
До выполнения                После выполнения
Память данных                Память данных
302h   01h                   302h  01h
ACC    04h  C=X              ACC   08h  C=0
```

ПРИМЕР 2

```
RPT #15
SUBC *
```

До выполнения

```
ARP     3
AR3     1000h
```

Память данных

```
1000h   07h
ACC     41h  C=X
```

После выполнения

```
ARP     3
AR3     1000h
```

Память данных

```
1000h   07h
ACC     20009h  C=1
```

SUBS – Вычитание из аккумулятора с запрещением знакового расширения

СИНТАКСИС

Прямая: SUBS dma

Косвенная: SUBS {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	0	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$

$0 \le n \le 7$

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:

(PC) + 1 -> PC

(ACC) – (dma) -> ACC

(dma) как беззнаковое 16-битное число

Зависит от OVM; Не зависит от SXM; Влияет на C и OV.

ОПИСАНИЕ

Содержимое ячейки памяти данных (dma) вычитается из содержимого аккумулятора (ACC) с подавлением знакового расширения. Результат сохраняется в аккумуляторе. Полученное содержимое аккумулятора - знаковое число. Бит C очищается, если вычитание вызывает заём, иначе бит C устанавливается в 1.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SUBS DAT2  ;(DP = 16, SXM = 1)
До выполнения                  После выполнения
Память данных                  Память данных
802h   F003h                   802h  F003h
ACC    F105h  C=X              ACC   102h  C=1
```

ПРИМЕР 2 SUBS * ;(SXM = 1)

До выполнения			После выполнения		
ARP	0		ARP	0	
AR0	310h		AR0	310h	
Память данных			Память данных		
310h	F003h		310h	F003h	
ACC	0FFF F105h	C=X	ACC	0FFF 0102h	C=1

SUBT - Вычитание из аккумулятора со сдвигом, определяемым TREG1 регистром

СИНТАКСИС

Прямая: SUBT dma

Косвенная: SUBT {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$

$0 \le n \le 7$

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:

(PC) + 1 -> PC

(ACC) - ((dma)**x** $2^{TREG1(3-0)}$) -> ACC

(dma) как беззнаковое 16-битное число

Если SXM = 1: (dma) знако-расширенное

Если SXM = 0: (dma) не знако-расширенное

Зависит от OVM, SXM и TRM; Влияет на C и OV.

ОПИСАНИЕ

Содержимое ячейки памяти данных (dma) сдвигается влево от 0 до 15 бит, как определено 4 младшими битами регистра TREG1, и вычитается из содержимого аккумулятора (ACC). Результат сохраняется в аккумуляторе. Знаковое расширение в значении dma контролируется битом SXM. Бит C очищается, если результат вычитания вызывает заём, иначе бит C устанавливается в 1.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+2†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
SUBT DAT127  ;(DP = 4)
До выполнения                 После выполнения
Память данных                 Память данных
2FFh   06h                    2FFh  06h
TREG1  08h                    TREG1 08h
ACC    FDA5h  C=X             ACC   F7A5h  C=1
```

ПРИМЕР 2 SUBT *

До выполнения
ARP 1
AR1 800h
Память данных
TREG1 08h
ACC 0h C=X

После выполнения
ARP 1
AR1 800h
Память данных
TREG1 08h
ACC FFFF FF00h C=0

TBLR – Чтение таблиц

СИНТАКСИС
Прямая: TBLR dma
Косвенная: TBLR {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	0	1	See Section 5.2						

ОПЕРАНДЫ
$0 \le dma \le 127$
$0 \le n \le 7$

ИСПОЛНЕНИЕ
Прямая и косвенная адресация:
(PC) + 1 -> PC
(PFC) -> MCS
(ACC(15-0) -> PFC

Если (счётчик повторов)≠ 0:
(pma, адресуемый через PFC) -> dma.
Изменение текущего AR и ARP как указано
(PFC) + 1 -> PFC
(счётчик повторов) – 1 -> счётчик повторов.
Иначе: (pma, адресуемый через PFC) -> dma.
Изменение текущего AR и ARP как указано
(MCS) -> PFC.

ОПИСАНИЕ
Содержимое ячейки памяти программ (pma) передаётся ячейке памяти данных (dma). Pma определяется содержимым младшего байта аккумулятора (ACCL) и dma определяется из инструкции. Инструкция TBLR в цикле RPT становится одноцикловой инструкцией с автоинкрементацией pma.

СЛОВА
1

ЦИКЛЫ

	Cycles for a Single Instruction			
Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	3	3	3	$3+p_{code}$
Source: SARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: External Destination: DARAM	$3+p_{src}$	$3+p_{src}$	$3+p_{src}$	$3+p_{src}+p_{code}$
Source: DARAM/ROM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: SARAM Destination:	3	3	3, 4†	$3+p_{code}$

Source: SARAM Destination: External	$3+p_{src}$	$3+p_{src}$	$3+p_{src}$, $4+p_{src}$†	$3+p_{src}+p_{code}$
Source: External Destination: DARAM/ROM	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+p_{code}$
Source: External Destination: SARAM	$4+d_{dst}$	$4+d_{dst}$	$4+d_{dst}$	$6+d_{dst}+p_{code}$
Source: External Destination: External	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$4+p_{src}+d_{dst}$	$6+p_{src}+d_{dst}+p_{code}$

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM/ROM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: External Destination: DARAM	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}+p_{code}$
Source: DARAM/ROM Destination: SARAM	n+2	n+2	n+2, n+4†	$n+2+p_{code}$

Cycles for a Repeat (RPT) Execution (Continued)

Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, 2n+2§	$n+2+p_{code}$, 2n‡
Source: External Destination: SARAM	$n+2+np_{src}$	$n+2+np_{src}$	$n+2+np_{src}$, $n+4+np_{src}$†	$n+2+np_{src}+p_{code}$
Source: DARAM/ROM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+4+nd_{dst}+p_{code}$
Source: SARAM Destination: External	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+2+nd_{dst}$	$2n+4+nd_{dst}+p_{code}$
Source: External Destination: External	$4n+np_{src}+nd_{dst}$	$4n+np_{src}+nd_{dst}$	$4n+np_{src}+nd_{dst}$	$4n+2+np_{src}+nd_{dst}+p_{code}$

ПРИМЕР 1

```
TBLR DAT6  ;(DP = 4)
```

До выполнения		После выполнения	
ACC	23h	ACC	23h
Память программ		Память программ	
23h	306h	23h	306h
Память данных		Память данных	
206h	75h	206h	306h

ПРИМЕР 2

```
TBLR *,AR7
```

До выполнения		После выполнения	
ARP	0	ARP	7
AR0	300h	AR0	300h
ACC	24h	ACC	24h
Память программ		Память программ	
24h	307h	24h	307h
Память данных		Память данных	
300h	75h	300h	307h

TBLW – Запись таблиц

СИНТАКСИС

Прямая: TBLW dma

Косвенная: TBLW {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	1	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	1	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$

$0 \le n \le 7$

ИСПОЛНЕНИЕ

Прямая и косвенная адресация:

(PC) + 1 -> PC

(PFC) -> MCS

(ACC(15-0) -> PFC

Если (счётчик повторов)≠ 0:

(pma, адресуемый через PFC) -> pma.

Изменение текущего AR и ARP как указано

(PFC) + 1 -> PFC

(счётчик повторов) – 1 -> счётчик повторов.

Иначе: (dma, адресуемый через PFC) -> pma.

Изменение текущего AR и ARP как указано

(MCS) -> PFC.

ОПИСАНИЕ

Содержание ячейки памяти данных (dma) передаётся в память программ (pma). Dma определяется инструкцией, pma определяет содержимым младшего байта аккумулятора (ACCL). Когда используют с RPT инструкцией, TBLW становится одно-цикловой инструкцией с автоинкремтацией pma.

СЛОВА

1

ЦИКЛЫ

	Cycles for a Single Instruction			
Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: SARAM Destination: DARAM	3	3	3	$3+p_{code}$
Source: External Destination: DARAM	$3+d_{src}$	3+dsrc	$3+d_{src}$	$3+d_{src}+p_{code}$
Source: DARAM Destination: SARAM	3	3	3, 4†	$3+p_{code}$
Source: SARAM Destination:	3	3	3, 4†	$3+p_{code}$

Source: SARAM Destination: External	$3+d_{src}$	$3+d_{src}$	$3+d_{src}$, $4+d_{src}$†	$3+d_{src}+p_{code}$
Source: SARAM Destination: DARAM	$4+p_{dst}$	$4+p_{dst}$	$4+p_{dst}$	$5+p_{dst}+p_{code}$
Source: External Destination: SARAM	$4+p_{dst}$	$4+p_{dst}$	$4+p_{dst}$	$5+p_{dst}+p_{code}$
Source: External Destination: External	$4+d_{src}+p_{dst}$	$4+d_{src}+p_{dst}$	$4+d_{src}+p_{dst}$	$5+d_{src}+p_{dst}+p_{code}$

	Cycles for a Single Instruction			
Operand	ROM	DARAM	SARAM	External Memory
Source: DARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: SARAM Destination: DARAM	n+2	n+2	n+2	$n+2+p_{code}$
Source: External Destination: DARAM	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}+p_{code}$
Source: DARAM Destination: SARAM	n+2	n+2	n+2, n+3†	$n+2+p_{code}$

	Cycles for a Single Instruction			
Operand	ROM	DARAM	SARAM	External Memory
Source: SARAM Destination: SARAM	n+2, 2n‡	n+2, 2n‡	n+2, 2n‡, 2n+1§	$n+2+p_{code}$, 2n‡
Source: External Destination: SARAM	$n+2+nd_{src}$	$n+2+nd_{src}$	$n+2+nd_{src}$, $n+3+nd_{src}$†	$n+2+nd_{src}+p_{code}$
Source: DARAM Destination: External	$2n+2+np_{dst}$	$2n+2+np_{dst}$	2n+2+npdst	$2n+3+np_{dst}+p_{code}$
Source: SARAM Destination: External	$2n+2+np_{dst}$	$2n+2+np_{dst}$	2n+2+npdst	$2n+3+np_{dst}+p_{code}$
Source: External Destination: External	$4n+nd_{src}+np_{dst}$	$4n+nd_{src}+np_{dst}$	$4n+nd_{src}+np_{dst}$	$4n+1+nd_{src}+np_{dst}+p_{code}$

ПРИМЕР 1 TBLW DAT5 ;(DP = 32)

До выполнения		После выполнения	
ACC	257h	ACC	257h
Память данных		Память данных	
1905h	4339h	1905h	4339h
Память программ		Память программ	
257h	306h	257h	4399h

ПРИМЕР 2 TBLW *

До выполнения		После выполнения	
ARP	6	ARP	6
AR6	1006h	AR6	1006h
ACC	258h	ACC	258h
Память данных		Память данных	
1006h	4340h	1006h	4340h
Память программ		Память программ	
258h	307h	258h	4340h

TRAP - Программное прерывание

ОПЕРАНДЫ Нет

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	0	0	0	1

ИСПОЛНЕНИЕ
```
(PC) + 1 -> stack
22h -> PC
```

Не зависит от INTM; не влияет на INTM.

ОПИСАНИЕ TRAP вызывает последовательность инструкций программы расположенной 22h. Текущий программный счётчик (PC) инкрементируется и сохраняется в стек. Адрес 22h загружается в программный счётчик. По адресу 22h может содержаться инструкцию перехода к управлению определённым TRAP. Инструкция TRAP не маскируется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
4	4	4	4+3p†

ПРИМЕР
```
TRAP ; контроль прохождения ячейки 22h памяти программ и
задвигание PC + 1 в стэк.
```

XC – Условный выполнение инструкции

СИНТАКСИС XC n , cond [, cond1] [,...]

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	N†	0	1	TP†		ZLVC†				ZLVC†			

Operand (n) value	Opcode (N) value
1	0
2	1

ОПЕРАНДЫ n = 1 или 2

Условия:

ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEO
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
TC = 1	TC
Низкий $\overline{\text{BIO}}$	BIO
Безусловный	UNC

ИСПОЛНЕНИЕ Если условие(я): исполнение следующих n инструкций.
Иначе: выполнение NOPs для следующих n инструкций.

Не влияет на статусные биты.

ОПИСАНИЕ Если n = 1 и условие выполняется, 1-словная инструкция следующая за XC выполняется. Если n = 2 и условие выполняется,следущие 2-словная инструкция или две 1-словных инструкции выполненяются. Не все комбинации условий выполняются. Если условия не выполняются, выполняется одно или два NOPs.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction			
ROM	DARAM	SARAM	External Memory
1	1	1	1+p†

ПРИМЕР

```
XC 1,LEQ,C
MAR *+
ADD DAT100
```

Если содержимое аккумулятора меньше или равно 0 и бит C установлен, предыдущий ARP изменяется перед исполнением инструкции ADD.

XOR – Исключающее ИЛИ аккумулятора с значением памяти данных

СИНТАКСИС
Прямая: XOR dma
Косвенная: XOR {индекс}[,новый ARP]
Длинная непосредственная: XOR #lk[,сдвиг]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	0	1	See Section 5.2						

Long immediate addressing with shift

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	0	1	SHFT †			
16-Bit Constant															

Long immediate addressing with shift of 16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	1	0	0	0	0	0	1	1
16-Bit Constant															

ОПЕРАНДЫ
0 ≤ dma ≤ 127
0 ≤ n ≤ 7
lk: 16-битная константа
0 ≤ сдвиг ≤ 15

ИСПОЛНЕНИЕ
Прямая и косвенная адресация:
(PC) + 1 -> PC
(ACC(15-0)) XOR (dma) -> ACC(15-0)
(ACC(31-16)) -> ACC(31-16)

Длинная непосредственная константа со сдвигом:
(PC) + 2 -> PC
(ACC(31-0)) XOR (lk **x** 2^{shift}) -> ACC(31-0)

Не влияет на C; не зависит от SXM (длинная непосредственная константа).

ОПИСАНИЕ
Если длинная непосредственная константа определена, константа сдвигается влево, как определено сдвиговым кодом, дополняется до 32 битов 0 в старшие и младшие биты и выполняется XOR с содержимым аккумулятора (ACC). Результат сохраняется в аккумуляторе. Если константа не определена, содержимое ячейки памяти данных (dma) XOR с содержимым младшего байта аккумулятора (ACCL). Результат сохраняется в ACCL и содержимое старшего байта аккумулятора (ACCH) не меняется.

СЛОВА
1 (прямая и косвенная адресация)
2 (длинная непосредственная адресация)

ЦИКЛЫ

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution (direct or indirect addressing)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
2	2	2	2+2p†

ПРИМЕР 1
```
XOR DAT127  ;(DP = 511)
До выполнения                  После выполнения
Память данных                  Память данных
FFFFh   F0F0h                  FFFFh  F0F0h
ACC     1234 5678h  C=X         ACC    1234 A688h  C=X
```

ПРИМЕР 2
```
XOR *+,AR0
До выполнения                  После выполнения
ARP     7                      ARP     0
AR7     300h                   AR7     301h
Память данных                  Память данных
300h    FFFFh                  300h    FFFFh
ACC     1234 F0F0h  C=X         ACC     1234 0F0Fh  C=X
```

ПРИМЕР 3
```
XOR #0F0F0h,4
ACC     1111 1010h  C=X         ACC     111E 1F10h  C=X
```

XORB – Исключающее ИЛИ буфера аккумулятора (ACCB) с аккумулятором

СИНТАКСИС XORB

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	0

ОПЕРАНДЫ нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
(ACC) XOR (ACCB) -> ACC

Не влияет на статусные биты.

ОПИСАНИЕ Выполняется операция исключающего ИЛИ содержимого аккумулятора (ACC) с содержимым буфера аккумулятора (ACCB). Результат сохраняется в аккумуляторе, содержимое буфера аккумулятора не меняется.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction			
ROM	DARAM	SARAM	External Memory
1	1	1	1+p†

Cycles for a Single Instruction			
ROM	DARAM	SARAM	External Memory
n	n	n	n+p†

ПРИМЕР 1 XORB

До выполнения		После выполнения	
ACCB	F0F0 F0F0h	ACCBh	F0F0 F0F0h
ACC	FFFF 0000h	ACC	0F0F F0F0h

XPL – Исключающее ИЛИ низкого непосредственного или DBMR с значением памяти данных

СИНТАКСИС

Прямая: XPL [#lk,] dma
Косвенная: XPL [#lk,]{индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	0	dma						

Indirect addressing with long immediate not specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	0	1	See Section 5.2						

Direct addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	0	0	dma						
16-Bit Constant															

Indirect addressing with long immediate specified

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	0	1	See Section 5.2						
16-Bit Constant															

ОПЕРАНДЫ

0 ≤ dma ≤ 127
0 ≤ n ≤ 7
l*k*: 16-битная константа

ИСПОЛНЕНИЕ

Длинная непосредственная не задана:
(PC) + 1 -> PC
(dma) XOR (DBMR) -> (dma)

Длинная непосредственная задана:
(PC) + 2 -> PC
(dma) XOR lk -> (dma)

Влияет на TC

ОПИСАНИЕ

Если длинная непосредственная константа определена, проводится операция исключающего ИЛИ (XOR) константы с содержимым ячейки памяти данных (dma). Если константа не определена, XOR содержимого dma с содержимым DBMR (динамическим битом регистра обращения). В обоих случаях результат тут же записывается обратно в dma. Содержимое аккумулятора (ACC) не изменяется. Если результат операции XOR нулевой, то устанавливается в 1 бит TC, иначе бит TC очищается.

СЛОВА

1 (Длинная непосредственная не задана)
2 (Длинная непосредственная задана)

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 3†	1+p
External	2+2d	2+2d	2+2d	5+2d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory

DARAM	n	n	n	n+p
SARAM	2n–2	2n–2	2n–2,	2n–2+p
External	4n–2+2nd	4n–2+2nd	2n+1†	4n+1+2nd+p

Cycles for a Single Instruction (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	2	2	2	2+2p
SARAM	2	2	2	2+2p
External	3+2d	3+2d	3+2d	6+2d+2p

Cycles for a Repeat (RPT) Execution (long immediate specified)

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n+1	n+1	n+1	n+1+2p
SARAM	2n–1	2n–1	2n–1,	2n–1+2p
External	4n–1+2nd	4n–1+2nd	2n+2†	4n+2+2nd+2p

ПРИМЕР 1
```
XPL #100h,DAT60 ;(DP = 0)
До выполнения                  После выполнения
Память данных                  Память данных
60h   01h                      60h  101h
```

ПРИМЕР 2
```
XPL DAT60 ;(DP = 0)
До выполнения                  После выполнения
DBMR   FFFFh                   DBMR    FFFFh
Память данных                  Память данных
60h    0101h                   60h     FEFEh
```

ПРИМЕР 3
```
XPL #1000h,*,AR6
До выполнения                  После выполнения
ARP     0                      ARP     6
AR0     300h                   AR0     300h
Память данных                  Память данных
300h    FF00h                  300h    EF00h
```

ПРИМЕР 4
```
XPL *-,AR0
До выполнения                  После выполнения
ARP     6                      ARP     0
AR6     301h                   AR6     300h
DBMR    FF00h                  DBMR    FF00h
Память данных                  Память данных
301h    EF00h                  301h    1000h
```

ZALR – Обнуление младшего слова аккумулятора, загрузка старшего слова аккумулятора значением памяти данных; с округлением

СИНТАКСИС

Прямая: ZALR dma
Косвенная: ZALR {индекс}[,новый ARP]

КОД КОМАНДЫ

Direct addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	0	dma						

Indirect addressing

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	1	See Section 5.2						

ОПЕРАНДЫ

$0 \le dma \le 127$
$0 \le n \le 7$

ИСПОЛНЕНИЕ

(PC) + 1 -> PC
8000h -> ACC(15-0)
(dma) -> ACC(31-16)
Не влияет на C

ОПИСАНИЕ

Содержимое ячейки памяти данных (dma) загружается в старший байт аккумулятора. 15 младших бит (биты 0-14) аккумулятора устанавливаются в 0, бит 15 ACCL устанавливается в 1.

СЛОВА

1

ЦИКЛЫ

Cycles for a Single Instruction

Operand	ROM	DARAM	SARAM	External Memory
DARAM	1	1	1	1+p
SARAM	1	1	1, 2†	1+p
External	1+d	1+d	1+d	2+d+p

Cycles for a Repeat (RPT) Execution

Operand	ROM	DARAM	SARAM	External Memory
DARAM	n	n	n	n+p
SARAM	n	n	n, n+1†	n+p
External	n+nd	n+nd	n+nd	n+1+p+nd

ПРИМЕР 1

```
ZALR DAT3 ;(DP = 32)
До выполнения                  После выполнения
Память данных                  Память данных
1003h   3F01h                  1003h  3F01h
ACC     0077 FFFFh  C=X        ACC    3F01 8000h  C=X
```

ПРИМЕР 2

```
ZALR *-,AR4
До выполнения                  После выполнения
ARP    7h                      ARP    4h
AR7    FF00h                   AR7    FEFFh
Память данных                  Память данных
FF00h  E0E0h                   FF00h  E0E0h
ACC    0010 7777h  C=X         ACC    E0E0 8000h  C=X
```

ZAP – Обнуление аккумулятора и регистра произведения

СИНТАКСИС ZAP

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	1

ОПЕРАНДЫ нет

ИСПОЛНЕНИЕ
(PC) + 1 -> PC
0 -> ACC
0 -> PREG

Не влияет

ОПИСАНИЕ Содержимое аккумулятора (ACC) и регистра результата (PREG) очищается.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p†

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p†

ПРИМЕР ZAP

До выполнения		После выполнения	
PREG	3F01 1111h	PREG	0000 0000h
ACC	77FF FF77h	ACC	0000 0000h

ZPR – Обнуление регистра произведения

СИНТАКСИС ZAP

КОД КОМАНДЫ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	0	1	0	1	1	0	0	0

ОПЕРАНДЫ нет

ИСПОЛНЕНИЕ (PC) + 1 -> PC
0 -> PREG

Не влияет

ОПИСАНИЕ Содержимое регистра результата (PREG) очищается.

СЛОВА 1

ЦИКЛЫ

Cycles for a Single Instruction

ROM	DARAM	SARAM	External Memory
1	1	1	1+p†

Cycles for a Repeat (RPT) Execution

ROM	DARAM	SARAM	External Memory
n	n	n	n+p†

ПРИМЕР ZPR

До выполнения	После выполнения
PREG 3F01 1111h	PREG 0000 0000h

4.2.4 Таблица кодов операций системы команд

Этот раздел содержит свод опкодов команд 1867ВЦ2Т, 1867ВЦ2АТ. Эти данные используются для 1867ВЦ2Т, 1867ВЦ2АТ. Свод ранжирован согласно выполняемой функции и расположен по алфавиту в каждой категории.

Следующие символы используются в таблице опкодов:

Символы	Смысл
A R X	Трехбитовое поле содержания дополнительного регистра (0-7).
B I T X	Четырехбитовое поле, задающее бит тестирования для команды BIT.
|||| ||||	Короткое непосредственное значение.
INTR#	Номер вектора прерывания.
S H F T	Величина сдвига.
N	Поле для команды XC, указывающее число команд (1 или 2 условно выполняемых)
T P	Два бита, используемые командами условного исполнения, для представления условий TC, NTC, BIO.
V E C T R	Ячейка вектора прерывания
Z L V C	Четырехбитовое поле, представляющее следующие условия: Z: ACC = 0 L: ACC < 0 V: Переполнение C: Перенос Условные команды используют два таких 4-битовых поля. Младшие 4 бита команды – поле 4-битовой маски. Единичный бит в маске показывает, что условие будет тестироваться. Второе 4-битовое поле (биты 4-7)- состояния условий, которые будут тестироваться. Например, при тестировании на ACC≥0, поля Z и L могут быть установлены,а поля V и C не установлены. Следующие 4 бита поля содержат состояние условий тестирования. Поле Z будет установлено для индикации тестирования на ACC = 0, а поле L – на ACC НЕ МЕНЬШЕ ЧЕМ 0. Условия, которые могут быть сформированы в этих 8-ми битах, представлены командами в описании BCND, CC,XC.
+ 1 слово	Указывает на двухсловные команды. Второе слово 16-ти битовое длинное непосредственное значение или 16-ти битовый адрес программной памяти для непосредственной адресации.

Таблица 4.12 – Свод опкодов

Команды, работающие с аккумулятором		
Описание	Мнемоника	Опкоды
1	2	3
Абсолютное значение ACC	ABS	1011 1110 0000 0000
Сложить ACCB с ACC с переносом	ADCB	1011 1110 0001 0001
Прибавить к ACC со сдвигом	ADD	0010 SHFT |AAA AAAA
Прибавить к млад. ACC корот.непосред.	ADD	1011 1000 |||| ||||
Прибавить к ACC длин.непосред.со сдвигом непосредственную со сдвигом	ADD	1011 1111 1001 SHFT + 1 слово

Продолжение таблицы 4.12

1	2	3
Прибавить к ACC со сдвигом на 16	ADD	0110 0001 |AAA AAAA
Прибавить к ACC с переносом	ADDC	0110 0000 |AAA AAAA
Прибавить ACCB к ACC	ADDB	1011 1110 0001 0000
Прибавить к младшему слову ACC с подавлением расширения знака	ADDS	0110 0010 |AAA AAAA
Прибавить к ACC со сдвигом, заданным TREG1	ADDT	0110 0011 |AAA AAAA
AND со значением данных	AND	0110 1110 |AAA AAAA
AND с ACC длинное непосредственное со сдвигом	AND	1011 1111 1011 SHFT + 1 слово
AND с ACC длинное непосредственное со сдвигом на 16	AND	1011 1110 1000 0001 + 1 слово
AND ACCB с ACC	ANDB	1011 1110 0001 0010
Барабанный сдвиг ACC вправо	BSAR	1011 1111 1110 SHFT
Дополнить до 2 ACC	CMPL	1011 1110 0000 0001
Запоминание ACC в ACCB если ACC > ACCB	CRGT	1011 1110 0001 1011
Запоминание ACC в ACCB если ACC < ACCB	CRLT	1011 1110 0001 1100
Загрузить ACC в ACCB	EXAR	1011 1110 0001 1101
Загрузить ACCB в ACC	LACB	1011 1110 0001 1111
Загрузить ACC со сдвигом	LACC	0001 SHFT |AAA AAAA
Загрузить ACC длинное непосредственное со сдвигом	LACC	1011 1111 1000 SHFT + 1 слово
Загрузить ACC со сдвигом на 16	LACC	0110 1010 |AAA AAAA
Загрузить младшее слово ACC с непосредственным	LACL	1011 1001 |||| ||||
Загрузить младшее слово ACC	LACL	0110 1001 |AAA AAAA
Загрузить ACC со сдвигом из TREG1	LACT	0110 1011 |AAA AAAA
Загрузить в ACCL содержимое отображаемого в память регистра	LAMM	0000 1000 |AAA AAAA
Изменить знак ACC	NEG	1011 1110 0000 0010
Нормализация ACC	NORM	1010 0000 |AAA AAAA
OR ACC с значением данных	OR	0110 1101 |AAA AAAA
OR с ACC длинное непосредственное со сдвигом	OR	1011 1111 1100 SHFT + 1 слово
OR с ACC длинное непосредственное со сдвигом на 16	OR	1011 1110 1000 0010 + 1 слово
OR ACCB с ACC	ORB	1011 1110 0001 0011
Циклический сдвиг ACC влево на 1 бит	ROL	1011 1110 0000 1100
Циклический сдвиг ACCB и ACC влево	ROLB	1011 1110 0001 0100
Циклический сдвиг ACC вправо на 1 бит	ROR	1011 1110 0000 1101
Циклический сдвиг ACCB и ACC вправо	RORB	1011 1110 0001 0101
Загрузить ACC в ACCB	SACB	1011 1110 0001 1110
Записать старшее слово ACC со сдвигом	SACH	1001 1SHF |AAA AAAA
Записать младшее слово ACC со сдвигом	SACL	1001 0SHF |AAA AAAA
Загрузить ACC в отображаемый в память регистр	SAMM	1000 1000 |AAA AAAA
Барабанный сдвиг вправо 0-16 бит ACC вправо как задано регистром TREG1	SATH	1011 1110 0101 1010
Барабанный сдвиг вправо 0-15 бит ACC вправо как задано регистром TREG1	SATL	1011 1110 0101 1011
Вычесть ACCB из ACC	SBB	1011 1110 0001 1000
Вычесть ACCB из ACC с займом	SBBB	1011 1110 0001 1001
Сдвиг ACC влево	SFL	1011 1110 0000 1001
Сдвиг ACC и ACCB влево	SFLB	1011 1110 0001 0110
Сдвиг ACC вправо	SFR	1011 1110 0000 1010
Сдвиг ACC и ACCB вправо	SFRB	1011 1110 0001 0111
Вычесть из ACC со сдвигом	SUB	0011 SHFT |AAA AAAA
Вычесть из ACC со сдвигом на 16	SUB	0110 0101 |AAA AAAA
Вычесть из ACC короткого непосредственного	SUB	1011 1010 |||| ||||

Продолжение таблицы 4.12

1	2	3
Вычесть из ACC длинного непосредственного со сдвигом	SUB	1011 1111 1010 SHFT + 1 слово
XOR с ACC длинным непосредственным со сдвигом на 16	XOR	1011 1110 1000 0011 + 1 слово
XOR ACCB с ACC	XORB	1011 1110 0001 1010
Обнулить младшее слово ACC и загрузить старшее с округлением	ZALR	0110 1000 |AAA AAAA
Обнулить ACC и PREG	ZAP	1011 1110 0101 1001
Инструкции, работающие с дополнительными регистрами и указателем страниц		
Описание	Мнемоника	Опкоды
Прибавить к AR короткий непосредственный	ADRK	0111 1000 |||| ||||
Сравнить AR с ARCR	CMPR	1011 1111 0100 01CM
Загрузить AR из адресных данных	LAR	0000 0ARX |AAA AAAA
Загрузить AR короткий непосредственный	LAR	1011 0ARX |||| ||||
Загрузить AR длинный непосредственный	LAR	1011 1111 0000 1ARX + 1 слово
Загрузить указатель страницы данных из адресных данных	LDP	0000 1101 |AAA AAAA
Загрузить указатель страницы данных непосредственно	LDP	1011 110| |||| ||||
Модифицировать Arn	MAR	1000 1011 |AAA AAAA
Сохранить Arn	SAR	1000 0ARX |AAA AAAA
Вычесть из AR короткий непосредственный	SBRK	0111 1100 |||| ||||
Команды PLU		
Описание	Мнемоника	Опкоды
AND DMBR или константу со значением из памяти данных	APL	0101 1010 |AAA AAAA
AND короткий непосредственный со значением из памяти данных	APL	0101 1110 |AAA AAAA + 1 слово
Сравнить DMBR со значением из памяти данных	CPL	0101 1011 |AAA AAAA
Сравнить данные с длинной непосредственной	CPL	0101 1111 |AAA AAAA + 1 слово
OR DMBR со значением из памяти данных	OPL	0101 1001 |AAA AAAA
OR длинной непосредственной со значением из памяти данных	OPL	0101 1101 |AAA AAAA + 1 слово
Записать длинный непосредственный в память данных	SPLK	1010 1110 |AAA AAAA + 1 слово
XOR DMBR или константу со значением из памяти данных	XPL	0101 1000 |AAA AAAA
XOR длинной непосредственной со значением из памяти данных	XPL	0101 1100 |AAA AAAA + 1 слово
T-регистр, P-регистр и операции умножения		
Описание	Мнемоника	Опкоды
Прибавить PREG к ACC	APAC	1011 1110 0000 0100
Загрузить старшее слово PREG	LPH	0111 0101 |AAA AAAA
Загрузить TREG0	LT	0111 0011 |AAA AAAA
Загрузить TREG0 и аккумулировать предыдущий результат	LTA	0111 0000 |AAA AAAA
Загрузить TREG0, аккумулировать предыдущий результат и записать данные	LTD	0111 0010 |AAA AAAA
Загрузить TREG0 и записать TREG в аккумулятор	LTP	0111 0001 |AAA AAAA
Загрузить TREG0 и вычесть предыдущий результат	LTS	0111 0100 |AAA AAAA
Умножить и аккумулировать	MAC	1010 0010 |AAA AAAA + 1 слово

Продолжение таблицы 4.12

1	2	3
Умножить и аккумулировать с перемещением данных	MACD	1010 0011 |AAA AAAA + 1 слово
Умножить и аккумулировать с источником указанными BMAR и DMOV	MADD	1010 1011 |AAA AAAA
Умножить и аккумулировать с источником адреса BMAR	MADS	1010 1010 |AAA AAAA
Умножить значение из памяти данных TREG0 раз	MPY	0101 0100 |AAA AAAA
Умножить TREG0 с непосредственным 13-и битовым значением	MPY	110| |||| |||| ||||
Умножить TREG0 с длинной непосредственной	MPY	1011 1110 1000 0000 + 1 слово
Умножить TREG0 с данными, добавить предыдущий результат	MPYA	0101 0000 |AAA AAAA
Умножить TREG0 с данными, ACC-PREG	MPYS	0101 0001 |AAA AAAA
Умножить беззнаковое значение из памяти данных TREG0 раз	MPYU	0101 0101 |AAA AAAA
Загрузить PREG в ACC	PAC	1011 1110 0000 0011
Вычесть PREG из ACC	SPAC	1011 1110 0000 0101
Записать старшее слово PREG	SPH	1000 1101 |AAA AAAA
Записать младшее слово PREG	SPL	1000 1100 |AAA AAAA
Задать режим сдвига PREG	SPM	1011 1111 0000 00PM
Данные ->TREG, TREG0 возвести в квадрат и аккумулировать предыдущий результат	SQRA	0101 0010 |AAA AAAA
Данные ->TREG, TREG0 возвести в квадрат ACC-PREG	SQRS	0101 0011 |AAA AAAA
Обнулить PREG	ZPR	1011 1110 0101 1000
Команды перехода		
Описание	Мнемоника	Опкоды
Безусловный переход с новой AR	B	0111 1001 1AAA AAAA + 1 слово
Безусловный переход с новой AR	BD	0111 1101 1AAA AAAA + 1 слово
Переход на адрес из ACC	BACC	1011 1110 0010 0000
Переход на адрес из ACC с задержкой	BACCD	1011 1110 0010 0001
Переход по ARn != 0 с новой AR	BANZ	0111 1011 1AAA AAAA
Переход по ARn != 0 с новой AR	BANZD	0111 1111 1AAA AAAA
Условный переход	BCND	1110 00TP ZLVC ZLVC + 1 слово
Условный переход с задержкой	BCNDD	1111 00TP ZLVC ZLVC + 1 слово
Вызов подпрограммы с адреса из ACC	CALA	1011 1110 0011 0000
Вызов подпрограммы с адреса из ACC с задержкой	CALAD	1011 1110 0011 1101
Вызов подпрограммы с новой AR	CALL	0111 1010 1AAA AAAA + 1 слово
Вызов подпрограммы с новой AR с задержкой	CALLD	0111 1110 1AAA AAAA + 1 слово
Условный вызов п/п	CC	1110 10TP ZLVC ZLVC + 1 слово
Условный вызов п/п с задержкой	CCD	1111 10TP ZLVC ZLVC + 1 слово
Программное прерывание	INTR	1011 1110 011I NTR#
Немаскируемое прерывание	NMI	1011 1110 0101 0010
Возврат из п/п	RET	1110 1111 0000 0000
Условный возврат	RETC	1110 11TP ZLVC ZLVC
Условный возврат с задержкой	RETCD	1111 11TP ZLVC ZLVC
Возврат из п/п с задержкой	RETD	1111 1111 0000 0000

Продолжение таблицы 4.12

1	2	3
Возврат с переключением контекста и глобальным разрешением прерываний	RETE	1011 1110 0011 1010
Возврат из прерывания	RETI	1011 1110 0011 1000
Программное прерывание	TRAP	1011 1110 0101 0001
Условное выполнение следующей одной или двух команд	XC	111N 01TP ZLVC ZLVC
Операции в/в и операции с памятью		
Описание	Мнемоника	Опкоды
Перемещение блока из памяти данных в память данных	BLDD	1010 1000 |AAA AAAA + 1 слово
Перемещение блока из памяти данных в память данных DEST длинной непосредственной	BLDD	1010 1001 |AAA AAAA + 1 слово
Перемещение блока из памяти данных в память данных с источником в BMAR	BLDD	1010 1100 |AAA AAAA
Перемещение блока из памяти данных в память программ с DEST в BMAR	BLDP	0101 0111 |AAA AAAA
Перемещение блока из памяти программ в память данных	BLPD	1010 0101 |AAA AAAA + 1 слово
Перемещение блока из памяти программ в память данных с источником в BMAR	BLPD	1010 0100 |AAA AAAA
Перемещение данных в памяти данных	DMOV	0111 0111 |AAA AAAA
Ввод данных из порта	IN	1010 1111 |AAA AAAA
Чтение отображаемого в память регистра	LMMR	1000 1001 |AAA AAAA + 1 слово
Вывод данных в порт	OUT	0000 1100 |AAA AAAA + 1 слово
Запись отображаемого в память регистра	SMMR	0000 1001 |AAA AAAA + 1 слово
Чтение таблицы	TBLR	1010 0110 |AAA AAAA
Запись таблицы	TBLW	1010 0111 |AAA AAAA
Команды управления		
Описание	Мнемоника	Опкоды
Тестирование бита	BIT	0100 BITX |AAA AAAA
Тестирование бита, определенного TREG2	BITT	0110 1111 |AAA AAAA
Сброс режима переполнения	CLRC	1011 1110 0100 0010
Сброс расширения знака	CLRC	1011 1110 0100 0110
Сброс HOLD режима	CLRC	1011 1110 0100 1000
Сброс TC бита	CLRC	1011 1110 0100 1010
Сброс переноса	CLRC	1011 1110 0100 1110
Сброс CNF бита	CLRC	1011 1110 0100 0100
Сброс INTM бита	CLRC	1011 1110 0100 0000
Сброс XF pin	CLRC	1011 1110 0100 1100
Простой до прерывания	IDLE	1011 1110 0010 0010
Загрузить регистр статуса 0	LST#0	0000 1110 |AAA AAAA
Загрузить регистр статуса 1	LST#1	0000 1111 |AAA AAAA
Нет операции	NOP	1000 1011 0000 0000
Вытолкнуть вершину стека в младшее слово ACC	POP	1011 1110 0011 0010
Вытолкнуть вершину стека в память данных	POPD	1000 1010 |AAA AAAA
Поместить младшее слово ACC в стек	PUSH	1011 1110 0011 1100
Повтор следующей инструкции определено значением памяти раз	RPT	0000 1011 |AAA AAAA
Повтор следующей инструкции определено длинной непосредственной раз	RPT	1011 1110 1100 0100 + 1 слово
Повтор следующей инструкции определено короткой непосредственной раз	RPT	1011 1011 |||| ||||

Продолжение таблицы 4.12

1	2	3
Повтор блока	RPTB	1011 1110 1100 0110 + 1 слово
Повтор следующей инструкции длинной несредственной и обнуление ACC/PREG	RPTZ	1011 1110 1100 0101 + 1 слово
Установить режим переполнения	SETC	1011 1110 0100 0011
Установить режим расширения знака	SETC	1011 1110 0100 0111
Установка HOLD режима	SETC	1011 1110 0100 1001
Установка TC бита	SETC	1011 1110 0100 1011
Установка переноса	SETC	1011 1110 0100 1111
Установка XF pin	SETC	1011 1110 0100 1101
Установка CNF бита	SETC	1011 1110 0100 0101
Установка INTM бита	SETC	1011 1110 0100 0001
Сохранить регистр статуса 0	SST#0	1000 1110 |AAA AAAA
Сохранить регистр статуса 1	SST#1	1000 1111 |AAA AAAA
Простой до прерывания - режим малого потребления энергии	IDLE2	1011 1110 0010 0011

4.3 Память

Общий объем адресуемой памяти 1867ВЦ2Т, 1867ВЦ2АТ составляет 224К (килобайта) 16-битных слов. Все пространство памяти разделено на 4 отдельных сегмента:

- 64К-слов программ;
- 64К-слов локальных данных;
- 64К-слов порты входа/выхода;
- 32К-слов глобальные данные.

Параллельная архитектура процессора 1867ВЦ2Т (1867ВЦ2АТ) позволяет выполнять три конкурирующие операции памяти в любом фиксированном машинном цикле: инструкцию выборки, операцию чтения, операцию записи.

Структура процессора 1867ВЦ2Т (1867ВЦ2АТ) основана на расширенной Гарвардской архитектуре, имеющей множественные пространства памяти, к которым можно обратиться по двум параллельным шинам. Это позволяет обратиться и к программе и данным одновременно. Две параллельных шины - шина программы (PB) и шина чтения данных (DB). Каждая шина обращается к различным областям памяти для операций с различными устройствами. Память процессора 1867ВЦ2Т (1867ВЦ2АТ) состоит из четыре индивидуально выбираемых областей: памяти программ, локальная памяти данных, глобальной память данных и портов ввода - вывода. Эти области памяти составляют адресный интервал слов 224 КБ. RAM, ROM, EPROM (программируемое ПЗУ), EEPROM, периферийные устройства с отображаемой памятью могут находиться в пределах любого из этих пространств внутри или вне кристалла. Область программ 64К-слова содержит наборы команд, преднозначенных для выполнения. 64К-слово локальной области данных сохраняет данные, используемые командами. Область данных глобальной переменной 32К-слов может совместно использовать данные с другими процессорами в пределах системы или просто служить дополнительным пространством данных. Интерфейсы области порта ввода-вывода 64К-слов на внешние периферийные устройства с отображаемой памятью и могут также служить дополнительным пространством для хранения данных. В пределах данного машинного цикла арифметико-логическое устройство может выполнить целых три параллельных операции с памятью.

Большая внутрикристальная память процессора 1867ВЦ2Т (1867ВЦ2АТ) обеспечивает высокие характеристики функционирования и интеграции системы. Эта память на кристалле включает ПЗУ в программной области, ОЗУ одинарного доступа (SARAM) в пространстве программ/данных и ОЗУ двойного доступа (DARAM) в пространстве программ/данных.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет 1056 слов DARAM, сконфигурированные в трех блоках и отображаемые в тех же самых адресах: блок 0 (B0) имеет 512 слов в адресе 0100h-02FFh в локальной памяти данных или FE00h-FFFFh в пространстве программ; блок 1 (B1) имеет 512 слов в адресе, 0300h-04FFh в локальной памяти данных; блок 2 (B2) имеет 32 слова в адресе, 0060h-007Fh в локальной памяти данных. DARAM может *читаться из* и *записываться в* в одном и том же машинном цикле.

При обращении к блоку SARAM требуется полный машинный цикл для чтения или записи. Однако центральный процессор может одновременно обращаться к разным блокам SARAM в течение одного машинного цикла.

ПЗУ в пространстве программ может быть загрузочным или замаскировано. Внутрикристалльное масочное ПЗУ постоянно находится в области программ в адресе 0000h и включает тест устройства (для внутреннего использования) и загрузочный код. ПЗУ включается или блокируется управляющим входом MP / $\overline{MC}$ при сбросе, или управляется битом MP/MC в регистре состояния режима процессора (PMST) после сброса.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) (см. рисунок 4.21) включает 2-х килобайтные слова загрузочного ПЗУ, 9 КБ слова программ/данных SARAM и 1056 слов DARAM. Загрузочное ПЗУ постоянно находится в области программ в интервале адресов 0000h-07FFh. 9 КБ слова SARAM могут быть отображены в пространство программ или данных и постоянно находиться в интервале адресов, 0800h-2BFFh в любой области.

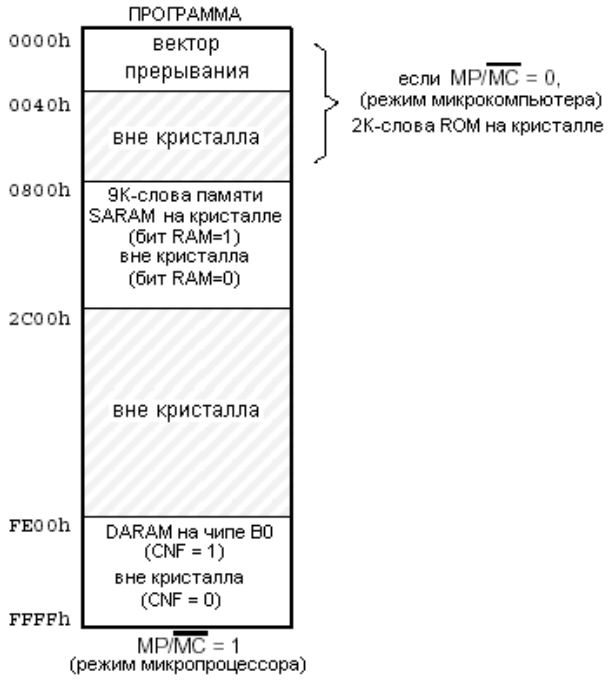


Рисунок 4.21 - Карта памяти 1867ВЦ2Т, 1867ВЦ2АТ

4.3.1 Память программ

Пространство памяти программ адресуется 64-мя килобайтами 16-битных слов и включает: внутрикристальное ПЗУ в области программ, SARAM в области программ и/или данных, и DARAM в области программ и/или данных. Программное обеспечение может конфигурировать эти ячейки памяти из карты адресов программ, чтобы те постоянно находились внутри (на кристалле) или снаружи (вне кристалла). Когда ячейки памяти отображаются в область программ, процессор 1867ВЦ2Т (1867ВЦ2АТ) автоматически обращается к ним, когда адреса лежат в пределах доступных границ. Когда ALU генерирует адрес вне доступной границы, то процессор 1867ВЦ2Т (1867ВЦ2АТ) автоматически организует внешний (вне кристалла) доступ. Таким образом, преимуществами работы с внутренней памятью (на кристалле) являются:

1) высокая производительность, поскольку более медленная внешняя память не имеет состояния ожидания;

2) низкая стоимость по сравнению с внешней памятью;

3) более высокая мощность по сравнению с внешней памятью.

Преимущество работы внешней памяти – способность обращаться к большому объему адресного пространства.

Характеристика процессора ЦОС 1867ВЦ2Т (1867ВЦ2АТ)

Внутрикристальная память (16-битные слова)			Порты ввода/вывода		Время цикла, нс		Тип корпуса
DARAM[1]	SARAM[2]	ROM	SER*	PARALLEL**	1867ВЦ2Т	1867ВЦ2АТ	
1056	9К	2К	2	64К	50/25	35/17,5	4229.132-3

[1] Dual-access RAM (DARAM)

[2] Single-access RAM (SARAM)

* SER - последовательный; один порт ввода и один порт вывода

** 16 из 64К параллельных портов входа/выхода, которые являются портами с управлением памяти

4.3.1.1 Конфигурируемость пространства программ

Память программ может находиться постоянно как внутри, так и вне кристалла. Конфигурация устройства 1867ВЦ2Т, 1867ВЦ2АТ такова, что при сбросе, устройство конфигурируется установкой уровня на входе $MP / \overline{MC}$. Если он установлен в единицу, то

устройство конфигурируется как микропроцессор и нет обращения к ПЗУ на кристалле. Если $MP/\overline{MC}$ установлен в нуль, то устройство сконфигурировано как микрокомпьютер, и внутрикристальное ПЗУ доступно.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) после сброса выбирает вектор в памяти программ по адресу 0000h; таким образом, если процессор работает как микрокомпьютер, то вызов происходит из памяти на кристалле. Если процессор работает как микропроцессор, выборка происходит из памяти вне кристалла. Как только программа запускается, конфигурация устройства может быть изменена, посредством регулирования или очистки бита $MP/\overline{MC}$ в PMST. Обратите внимание, что переключатель $MP/\overline{MC}$ выбирается только при сбросе. Следующая инструкция исключает память (ПЗУ) из области программы, путем установки значения бита $MP/\overline{MC}$ в PMST равным 1:

```
OPL*8, PMST; удаление ROM из области программы.
```

При сбросе SARAM и 512 слов DARAM блока B0 – не размещены в пространстве программ. Сделать SARAM областью программ возможно, устанавливая бит оперативной памяти (ОЗУ) в PMST. Когда разряд оперативной памяти установлен, ячейки оперативной памяти становятся областбю программ. Вы делаете доступным блок B0 DARAM в программном пространстве (адресный интервал FE00h-FFFFh), устанавливая бит CNF в ST1.

Следующие команды отображают SARAM и DARAM блоки в область программ, устанавливая соответствующий разряд в регистрах:

```
OPL     *010h, PMST    ; Карта памяти одно-портового (single access) доступа в
                         области программы.
SETC  CNF              ; Отобразите B0, чтобы программировать пространство{*пробел*}.
```

Таблица 4.13 показывает доступные конфигурации памяти программ процессора 1867ВЦ2Т (1867ВЦ2АТ). Обратите внимание, что все адреса заданы в шестнадцатеричной системе исчисления.

Таблица 4.13 - Управление конфигурацией памяти программ на 1867ВЦ2Т, 1867ВЦ2АТ

Значение бита			ROM (2К-слова)	SARAM (9К-слова)	DARAM B0 (512-слова)	Вне кристалла
CNF	RAM	MP / $\overline{MC}$				
0	0	0	0000-07FF	Вне крист.	Вне крист.	0800-FFFF
0	0	1	вне крист.	Вне крист.	Вне крист.	0000-FFFF
0	1	0	0000-07FF	0800-2BFF	Вне крист.	2C00-FFFF
0	1	1	вне крист.	0800-2BFF	Вне крист.	0800-FDFF
1	0	0	0000-07FF	Вне крист.	FE00-FFFF	0000-FDFF
1	0	1	вне крист.	Вне крист.	FE00-FFFF	2C00-FDFF
1	1	0	0000-07FF	0800-2BFF	FE00-FFFF	0000-07FF
1	1	1	вне крист.	0800-2BFF	FE00-FFFF	2C00-FDFF

4.3.1.2 Карта адресов памяти программ

Векторы прерывания обращаются в пространство программ. Эти векторы задаются программно, т.е. процессор, получая внутреннее прерывание, загружает счетчик программ (PC) с адресом прерывания и выполняет код, на который указывает вектор. В каждом векторном местоположении зарезервированы два слова для команды перехода к соответствующей программе обработки прерывания (ISR). В таблице 4.14 приведен список адресов векторов прерывания после сброса.

Таблица 4.14 - Адресация векторов прерывания

Имя	Расположение		Приоритет	Выполняемая функция
	Dec	Hex		
1	2	3	4	5
$\overline{\text{RS}}$	0	0	1 (высший)	Внешний немаскируемый сигнал сброса
$\overline{\text{INT1}}$	2	2	3	Внешнее пользовательское прерывание #1
$\overline{\text{INT2}}$	4	4	4	Внешнее пользовательское прерывание #2
$\overline{\text{INT3}}$	6	6	5	Внешнее пользовательское прерывание #3
TINT	8	8	6	Внутреннее прерывание таймера
RINT	10	A	7	Принятое последовательным портом прерывание
XINT	12	C	8	Переданное последовательным портом прерывание
TRNT	14	E	9	Полученное TDM портом прерывание
TXNT	16	10	10	Переданное TDM портом прерывание
$\overline{\text{INT4}}$	18	12	11	Внешнее пользовательское прерывание #4
——	20-23	14-17	N/A	Зарезервировано
HINT	24	18	——	Не используется
——	26-33	1A-21	N/A	Зарезервировано
TRAP	34	22	N/A	Программное (внутреннее) прерывание
$\overline{\text{NMI}}$	36	24	2	Немаскируемое прерывание
——	38-39	26-27	N/A	Зарезервировано для эмуляции и тестов
——	40-63	28-3F	N/A	Программное прерывание

При сбросе вектор прерывания отображается по адресу 0000h в пространстве программ. Однако вектор прерывания может быть повторно отображен к началу любой страницы 2К-слов в области программ после сброса. Для этого загружается разряд указателя вектора прерывания (IPTR) в PMST с соответствующим начальным адресом

страницы 2К-слов. После загрузки IPTR любой пользовательский вектор прерывания отображается в новой странице 2К-слов. Например:

```
OPL*05800h, PMST   ; перераспределение векторов для старта с адреса 5800h.
```

В этом примере векторы прерывания перемещаются в начало пространства программ вне кристалла в адрес 5800h. Любое последующее прерывание (если не произойдет сброс) выберет вектор прерывания от этого нового местоположения. Например, если после загрузки IPTR, происходит $\overline{\text{INT2}}$ (см. таблицу 4.14), то вектор программы обработки прерывания будет выбран по адресу 5804h в пространстве программ, а не по адресу 0004h. Эта особенность позволяет перемещать желаемые векторы из аппаратного загрузчика (ПЗУ начальной загрузки) и затем удалять ПЗУ из карты памяти. Как только системный код загружен в систему от резидентного загрузчика в ПЗУ, приложение перезагружает IPTR со значением, указывающим на новые векторы. В приведенном выше примере команда OPL используется, чтобы изменить IPTR биты в PMST. Этот пример предполагает, что IPTR биты в настоящее время очищены. Чтобы гарантировать правильное установленное значение IPTR, необходимо чтобы биты были очищены прежде, чем эта команда будет выполнена.

4.3.1.3 Адресация памяти программ

Пространство памяти программ содержит инструкции. Оно так же может содержать таблицу данных и непосредственных операндов.

К памяти программ обращаются только с помощью PAB. Адрес для этой шины генерируется PC, когда осуществляется доступ к инструкциям и длинным прямым операндам.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) выбирает команды, помещая PC в PAB и считывая соответствующее местоположение в памяти программ. В то время чтения инструкции, PC увеличивается для следующей выборки. Если последовательное приращение адреса программы нарушается (например, ветвление, вызов, возврат, прерывание или повторение блока), соответствующий адрес загружается в PC. PC также загружается, когда операнды выборку из памяти программ, это происходит, когда устройство читает из (TBLR) или пишет (TBLW) таблицы, когда происходит передача данных в (BLPD) или из (BLDP) пространства данных, или когда используется шина программ, чтобы выбрать второй сомножитель (MAC, MACD, MADS, и MADD).

Данные, используемые как операнды команд, получаются в одном из следующих способов адресации:

- режим прямой адресации;
- косвенный способ адресации;
- короткий прямой способ адресации;
- длинный прямой способ адресации;
- режим адресации через специальные регистры;
- адресации регистров, картированных в памяти.

4.3.2 Локальная память данных

Пространство локальной памяти данных процессора 1867ВЦ2Т (1867ВЦ2АТ) адресует до 64 КБ 16-разрядных слов. Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет 1056 слов DARAM, SARAM может быть размещён в пространстве памяти данных с помощью установки бита OVLY в PMST. Вы можете использовать программное обеспечение, чтобы конфигурировать ячейки памяти, постоянно находящиеся на кристалле или вне кристалла локальной таблицы адресов данных.

Когда ячейки памяти размещены в локальной области данных, процессор 1867ВЦ2Т (1867ВЦ2АТ) автоматически обращается к ним, тогда адресация происходит в пределах их границ. Когда ALU генерирует адрес вне границ внутрикристального ОЗУ, процессор 1867ВЦ2Т (1867ВЦ2АТ) автоматически генерирует внешний доступ (вне кристалла). Преимущества работы с внутренней памятью:

1) Более высокая производительность, поскольку состояния с нулевым временем ожидания требуются для более медленного внешнего блока памяти.
2) Более высокая производительность из-за более эффективной конвейерной операции.
3) Стоимость ниже, чем у внешней памяти.
4) Более низкая потребляемая мощность, чем у внешней памяти.

Преимущество работы с внешней памятью - способность обратиться к большему объему адресного пространства.

4.3.2.1 Конфигурируемость локального пространства данных

Локальная память данных может располагаться как внутри, так и вне кристалла. При сбросе процессор 1867ВЦ2Т (1867ВЦ2АТ) конфигурирует карту 1056 слов DARAM в локальной области данных.

Блок DARAM B0 может быть реконфигурирован в области программ установкой бита CNF в ST1. SARAM может быть отображен в пространстве данных установкой разряда OVLY в PMST.

4.3.2.2 Карта адреса локальной памяти данных

64К слов локальной памяти данных включают в себя картируемые в памяти регистры. Этот регистры расположены на нулевой странице данных. Эта страница имеет пять секций модулей регистров: регистр центрального процессора, регистр периферии, область, зарезервированная для тестирования/эмуляции, пространство ВВОДА/ВЫВОДА, сверхбыстрая оперативная память.

Регистры центрального процессора могут позволять доступ к с нулевым ожиданием. К части из этих регистров можно обратиться только через шину данных – например, используя вспомогательный модуль арифметики регистра при помощи команды LAR.

Регистры периферии производят управление, данные регистров используются в периферийных устройствах. Взаимодействие с этими регистрами осуществляется по специально выделенной шинной структуре, называемой TI-шиной. При обращении они требуют одно состояние ожидания.

Область, зарезервированная под тестирование/эмуляцию, используется системой тестирования и эмуляции для специальной информационной передачи.

Пространство ВВОДА/ВЫВОДА обеспечивает доступ к 16 словам пространства ВВОДА/ВЫВОДА (команды, отличные от IN и OUT) через возможные доступные состояния адресации внутри пространства данных. Например, команда SAMM может записаться в порт отображаемой памяти как выполненная команда OUT. Внешний интерфейс функционирует так, как будто команда OUT уже выполнена. Порт адреса постоянно находится вне кристалла и подчинен внешним состояниям ожидания. Они также затрагиваются программным генератором состояния ожидания, подобно любому другому порту, неотображенному в памяти.

Сверхбыстрая оперативная память блока B2 DARAM может использоваться для удержания верхних переменных так, пока большие блоки оперативной памяти не будут фрагментированы. Такой блок оперативной памяти обеспечивает операции двойного доступа и может обращаться как к катрированным в памяти регистрам, так к пространству данных.

4.3.2.3 Адресация памяти локальных данных

Создание адреса пространства локальных данных контролируется декодированием текущей команды. Память локальных данных читается через шину адресации данных в команде с одним операндом памяти и через шину адресации команд в команде со вторым операндом памяти данных. Однако адреса памяти данных могут быть созданы одним из следующих способов:

- режим прямой адресации;
- режим косвенной адресации;
- длинный непосредственный способ адресации;
- способ адресации с помощью специальных регистров;
- способ адресации регистров, картированных в памяти.

4.3.3 Глобальная память

Для многопроцессорного применения процессор 1867ВЦ2Т (1867ВЦ2АТ) может определять место пространству памяти глобальных данных и связываться с этим пространством через шину запроса и контрольные сигналы READY. Эта способность может быть использована для расширения карты адреса памяти данных при помощи наложения адресного пространства.

Такая глобальная память может использоваться более чем одним процессором, доступ к ней должен быть разрешен. Когда глобальная память используется, пространство адресов процессора разделено на глобальные и локальные участки. Локальная часть используется процессором для исполнения его индивидуальных функций, и глобальная часть используется для соединения с другими процессорами. Это облегчает многопроцессорную обработку данных, в которой данные передаются между двумя или более процессорами. В отличие от прямого доступа к памяти между двумя процессорами, чтение и запись в глобальной памяти не требуют приостановления работы другого процессора.

4.3.3.1 Конфигурируемость глобальной памяти

Регистр распределения глобальной памяти (GREG) определяет часть памяти данных процессора как глобальную внешнюю память. Восьмибитовый GREG отображается в памяти данных по адресу 05h и соединен с восемью младшими разрядами внутренней шины данных. Верхние 8 битов не используются и считываются как 1.

Содержимое GREG определяет размер (между 256 и 32К слов) пространства глобальной памяти. Действительный объем GREG и соответствующие пространства глобальной и локальной памяти приведены в таблице 4.15.

Таблица 4.15 - Конфигурация глобальной памяти данных

Уровень GREG	Локальная память		Глобальная память	
	область	# слов	область	# слов
0000 00XX	0000-FFFF	65 536	-	0
1000 0000	0000-7FFF	32 768	8000-FFFF	32 768
1100 0000	0000-BFFF	49 152	C000-FFFF	16 384
1110 0000	0000-DFFF	57 344	E000-FFFF	8192
1111 0000	0000-EFFF	61 440	F000-FFFF	4096
1111 1000	0000-F7FF	63 488	F800-FFFF	2048
1111 1100	0000-FBFF	64 512	FC00-FFFF	1024
1111 1110	0000-FDFF	65 024	FE00-FFFF	512
1111 1111	0000-FEFF	65 280	FF00-FFFF	256

4.3.3.2 Адресация глобальной памяти

Когда адрес памяти данных, прямой или косвенный, соответствует адресу глобальной памяти данных (как это описано для GREG), BR установлен низким с DS для того, чтобы показать начало доступа процессора к глобальной памяти. Внешняя логика управления глобальной памятью, выставляет READY, появление которого ожидает процессор 1867ВЦ2Т (1867ВЦ2АТ). Длина цикла памяти контролируется сигналом READY. Кроме того, для расширения временного доступа более медленной внешней памяти может использоваться генератор состояния ожидания. Режим генератора состояния

ожидания для соответствующих адресов в пространстве глобальной памяти данных, соответствует режиму для этих же адресов локальной памяти.

4.3.4 Пространство ввода/вывода

Процессор 1867ВЦ2Т (1867ВЦ2АТ) поддерживает адресное пространство ВВОДА/ВЫВОДА в 64К 16-разрядного параллельного входа и портов выхода. Порты ВВОДА/ВЫВОДА позволяют доступ к периферии, обычно используемой в DSP-применении (кодер-декодеров, аналогово-цифровых конвертеров, цифро-аналоговых конвертеров). В этом разделе описана адресация портов ВВОДА/ВЫВОДА и связь с внешними устройствами через порты ВВОДА/ВЫВОДА.

4.3.4.1 Адресация портов ввода/вывода

Доступ к внешним параллельным портам ввода/вывода осуществляется по тем же шинам данных и адреса, что и доступ к внешней памяти программ/данных. Все 64К портов ввода/вывода доступны через команды IN и OUT, как это показано в следующем примере:

```
IN DAT7, 0FFFEh; Считывание данных в память данных из внешнего устройства в
                 порт 65534.
OUT DAT7, 0FFFFh; Запись данных из памяти данных во внешнее устройство в порт
                  65535.
```

16 разрядов портов ввода/вывода картированы в памяти по адресу 50h-5Fh памяти данных. Порты ввода/вывода могут быть доступны через команды IN и OUT, которые читают или записывают местоположение в пространстве памяти данных.

Время доступа к портам ввода/вывода может изменяться через регистры состояния ожидания (IOWSR и CWSR). Разряд BIG в регистре CWSR определяет, как разделено пространство ввода/вывода.

4.3.5 Прямой доступ к памяти

Процессор 1867ВЦ2Т (1867ВЦ2АТ) поддерживает мультипроцесорнай режим, которое требует прямого доступа (DMA) к внешней памяти или внутрикристальному ОЗУ однакратного доступа. Особенность прямого доступа к памяти может использоваться для многопроцессорной обработки, временно останавливая работу одного или нескольких процессоров при выполнении другим процессором операций чтения или записи из локальной внешней памяти или внутренней ОЗУ одинарного доступа. Доступ к внешней

памяти может контролироваться через сигналы $\overline{\text{HOLD}}$ и $\overline{\text{HOLDA}}$, а доступ к внутреннему ОЗУ через сигналы $\overline{\text{HOLD}}$, $\overline{\text{HOLDA}}$, R/$\overline{\text{W}}$, $\overline{\text{STRB}}$, $\overline{\text{BR}}$, $\overline{\text{IAQ}}$.

4.3.5.1 Прямой доступ в схеме «главный-подчиненный»

Многопроцессорные системы обычно используют схему «ведущий-ведомый». Головной процессор может инициализировать ведомый процессор через загрузку программы в память ведомого процессора и/или может обеспечивать ведомый процессор необходимыми данными через использование внешней памяти для выполнения задачи.

В типичной схеме прямого доступа головной процессор может быть универсальным центральным процессором, другим процессором 1867ВЦ2Т (1867ВЦ2АТ) или даже управляющим устройством, как это показано на рисунке 4.22.

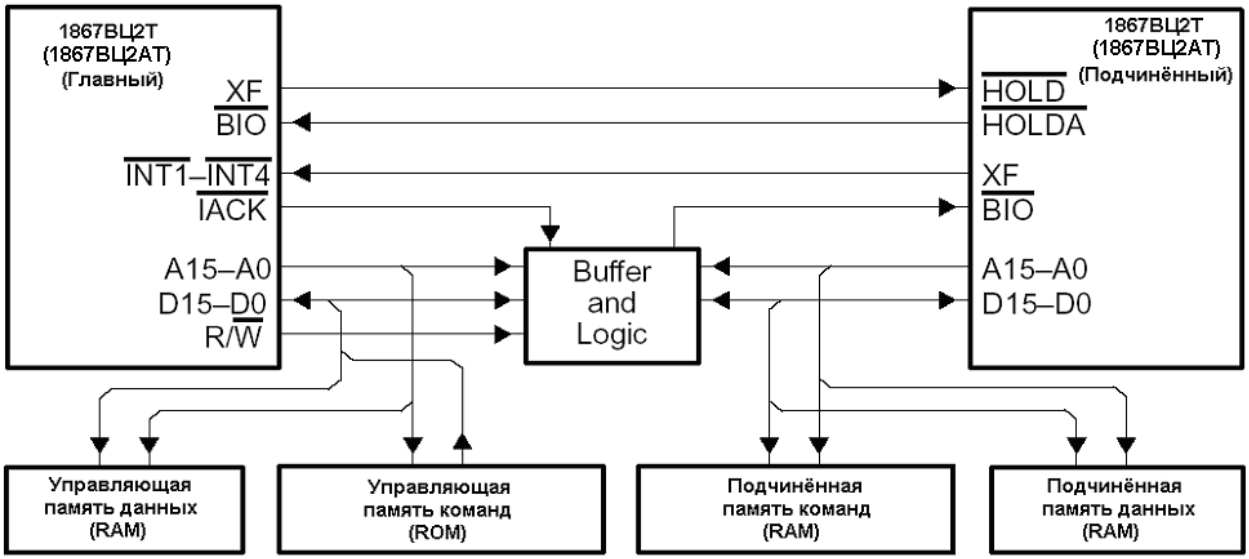


Рисунок 4.22 - Прямой доступ в схеме «главный-подчиненный»

Ведущий процессор 1867ВЦ2Т (1867ВЦ2АТ) полностью контролирует внешнюю память ведомого процессора, устанавливая сигнал $\overline{\text{HOLD}}$ ведомого процессора низким через внешний вывод управляющего процессора. Это заставляет подчиненный процессор устанавливать свои линии адреса, данных, линии управления в третье состояние.

Когда ведущий процессор, Когда ведомый процессор устанавливает $\overline{\text{HOLDA}}$, ведущий процессор получает контроль над шинами ведомого процессора Этот сигнал может быть соединен с выводом управляющего процессора $\overline{\text{BIO}}$. XF вывод подчиненного процессора может указывать управляющему процессору момент, когда подчиненный

процессор заканчивает выполнение своей задачи и нуждается в перепрограммировании или требует дополнительные данные для продолжения процесса. В подчиненном процессоре, состоящем из нескольких подобных процессоров, приоритет выполнения задачи определяется соединением сигнала XF с соответствующим $\overline{INT1}$ - $\overline{INT4}$ выводом управляющего процессора. Внешний шинный интерфейс подчиненного процессора устанавливается в третье состояние, когда устанавливается сигнал $\overline{HOLDA}$. Как только сигнал $\overline{HOLDA}$ становится активным, вывод $\overline{IAQ}$ не указывает команду захвата. Пока $\overline{HOLDA}$ активен и центральный процессор находится в режиме удержания (HM=0), центральный процессор продолжает кодирование из внутренней памяти (внутренняя ПЗУ или ОЗУ одинарного/двойного доступа). Если центральный процессор не находится в состоянии ожидания, то он останавливает внутреннее выполнение.

4.3.5.2 Внешний прямой доступ

Процессор 1867ВЦ2Т (1867ВЦ2АТ) также позволяет доступ внешним устройствам к внутреннему ОЗУ одинарного доступа через механизм, называемый внешним прямым доступом. Внешний прямой доступ требует следующие сигналы:

A(15-0) – Адресные входы, если $\overline{HOLDA}$ и $\overline{BR}$ имеют низкий уровень.

$\overline{BR}$ – Сигнал запроса шины низкий по внешнему управлению в состоянии удержания при запросе доступа.

D(15-0) – Данные прямого доступа.

$\overline{HOLD}$ – Внешний запрос для контроля над адресами, данными и контрольными линиями.

$\overline{HOLDA}$ – Указание внешней цепи, что адреса памяти, данные и контрольные линии находятся в третьем состоянии, разрешая внешний доступ.

$\overline{IAQ}$ – Запрос подтвержденного $\overline{BR}$ для доступа, пока $\overline{HOLDA}$ низок.

$R / \overline{W}$ – Сигнал «чтение/запись» указывает шине данных команду для прямого чтения (низкий) и прямой записи (высокий).

$\overline{STRB}$ – Когда $\overline{IAQ}$ и $\overline{HOLDA}$ низки, $\overline{STRB}$ выбирает доступ памяти и определяет его длительность.

Для доступа к внутреннему SARAM управляющий процессор должен контролировать процессор 1867ВЦ2Т (1867ВЦ2АТ). Управляющий процессор вызывает DMA передачу, устанавливая HOLD низким. Процессор 1867ВЦ2Т (1867ВЦ2АТ) отвечает, устанавливая $\overline{\text{HOLDA}}$. Управляющий процессор получает контроль над шиной и доступ к SARAM, устанавливая BR низким. Процессор 1867ВЦ2Т (1867ВЦ2АТ) отвечает, устанавливая $\overline{\text{IAQ}}$ низким при подтверждении доступа. Как только предоставлен доступ, управляющий процессор направляет $R/\overline{W}$ сигнал для выявления указания передачи. При прямой записи управляющий процессор должен управлять линиями данных и адресов для чтения. При прямом чтении управляющий процессор должен управлять линиями данных и адресов, а также фиксировать (защелкивать) данные. Каждый доступ к памяти (чтение или запись) выбирается при низком STRB. Декодирование адресов прямого доступа включает в себя только адресные линии A13-A0 (A14 и A15 игнорируются). Таблица 4.16 показывает области адресов во время прямого доступа.

Таблица 4.16 - Адресные области для внутрикристальной SARAM при прямом доступе

Устройство	SARAM (слова)	Шина данных	Область адреса
1867ВЦ2Т, 1867ВЦ2АТ	9K	A15-A14 игнорируются, A13-A0 используются	0000-2BFF 4000-6BFF 8000-ABFF C000-EBFF

Прямой доступ к процессору 1867ВЦ2Т (1867ВЦ2АТ) и запись в 01h задействуют вторую память из SARAM. Кроме того, запись по адресу 4001h в процессоре 1867ВЦ2Т (1867ВЦ2АТ) эквивалентна записи в адреса 01h, 8001h, C001h, в то время как линии A14 и A15 игнорируются.

Обратить внимание, что внешние параллельные интерфейсные сигналы асинхронно заблокированы во время сброса, следовательно, при этом прямой доступ к памяти не поддерживается.

4.3.6 Управление памятью

Программируемая карта памяти процессора 1867ВЦ2Т (1867ВЦ2АТ) может быть изменена для некоторых способов применения. Команды обеспечивают интегрирование

памяти устройства в карту системной памяти. Примеры перемещения и конфигурирования памяти приведены в данном разделе.

4.3.6.1 Перемещения «память-память»

Эффективно используют пространство памяти процессора 1867ВЦ2Т (1867ВЦ2АТ) следующие команды для перемещения блоков данных и команд, передачи слов и функции перемещения данных.

- Команды перемещения блоков данных и команд:
  - команда BLDD перемещает блок внутри памяти данных;
  - команда BLDP перемещает блок из памяти данных в память программ;
  - команда BLPD перемещает блок из памяти программ в память данных.
- Команды передачи слов данных и команд:
  - команда чтения таблицы (TBLR) считывает слова из памяти программ в память данных;
  - команда записи таблицы (TBLW) считывает слова из памяти данных в память программ.
- Команда перемещения данных (DMOV) разрешает доступ к данным и операциям.

Для команд перемещения данных один адрес получается из генератора адреса данных, в то время как другие адреса получаются из продолжительной непосредственной константы или из BMAR. Когда использование происходит с повторяющимися указаниями (RPT или RPTZ), эти указания эффективно выполняют перемещение блока из внутренней памяти во внешнюю.

Команда DMOV, реализованная внутри данных ОЗУ, эквивалентна той же команде процессора 1867ВМ2. Команда DMOV копирует слова из памяти данных, к которой непосредственно обращаются в настоящее время и расположенной во внутренней ОЗУ, в последующую, более старшую область, в то время как данные из адресованной области используются в том же самом цикле (например, посредством ALU). Операция ARAU может быть представлена в том же самом цикле, когда используется косвенная адресация. Команда DMOV может осуществлять алгоритмы, которые используют z^{-1} операции задержки, такие как свертывание и цифровая фильтрация.

Команда DMOV гораздо более эффективна, когда управление осуществляется в ОЗУ двойного доступа. Когда управление находится в ОЗУ одинарного доступа, DMOV требуется дополнительный цикл. Использование команды DMOV происходит через следующие команды:

- LTD загружает TREGO и суммирует результат с перемещением данных;
- MACD мультиплецирует и суммирует с перемещением данных;
- MADD мультиплецирует и суммирует с перемещением данных коэффициентный адрес, содержащийся в BMAR.

4.3.6.2 Перемещение блока памяти

Команда BLDD передает данные следующими способами:

- из внешней памяти данных во внешнюю память данных;
- из внешней памяти данных во внутреннюю память данных;
- из внутренней памяти данных во внутреннюю память данных;
- из внутренней памяти данных во внешнюю память данных.

Следующий пример показывает, как использовать команду BLDD при перемещении внешних данных (например, таблица коэффициентов) во внутренний DARAM блока B1:

```
*
* Эта стандартная программа использует команду BLDD для перемещения
   внешних данных во внутреннюю память данных.
MOVED LACC #8000h
SAMM BMAR      ; BMAR содержит источник адреса в памяти данных
LAR AR7,#300h  ; AR7 содержит dest адреса в памяти данных
MAR *,AR7      ;ARP = AR7
RPT #511    ; Переместить 512 значений из памяти данных в память данных
              блока B1
BLDD BMAR,*+
RET.
```

4.3.6.2.1 Перемещение данных из памяти данных в память программ

Команды BLDP и TBLW передают данные памяти программ следующими способами:

- из внешней памяти данных во внешнюю память программ;
- из внешней памяти данных во внутреннюю память программ;
- из внутренней памяти данных во внутреннюю память программ;
- из внутренней памяти данных во внутреннюю память программ.

Для системы, имеющей внешнюю память данных, но не имеющей внешней памяти программ, можно использовать команду BLDP для перемещения дополнительных блоков кодов во внутреннюю память программ. Следующие примеры показывают, как использовать команду BLDP для перемещения внешних данных во внутреннюю память программ.

Можно использовать также команду TBLW при передаче памяти данных к памяти программ. Команда TBLW отличается от команды BLDP, в которой сумматор содержит особый адрес памяти программ. Это дает определенные особенности при расчетах, которые идут быстрее, чем предыдущие, в размещении блока данных в памяти программ. Примеры показывают, как использовать команду TBLW при перемещении внешних данных во внутреннюю память программ.

Перемещение внешних данных во внутреннюю память программ при помощи команды BLDP:

```
*
* Это стандартный пример использования команды BLDP для перемещения внешних
   данных во внутреннюю память программ. Эта команда может также
   использоваться при загрузке в операционной системе ОЗУ из внешней памяти
   данных.
*
MOVEDP LACC #2000h
SAMM BMAR        ; BMAR содержит dest. адреса в памяти команд (1867ВЦ2Т,
                   1867ВЦ2АТ)
LAR AR7,#0F000h ; AR7 содержит источник адресов в памяти данных
MAR *,AR7        ; ARP=AR7
RPT #1023        ; Перемещает 1k значений из памяти данных в память команд
BLDP *+
RET
```

Перемещение внешних данных во внутреннюю память программ при помощи команды TBLW:

```
*
* Это стандартный пример использования команды TBLW при перемещении памяти
данных в память программ.
* Эта стандартная программа должна содержать предназначенные адреса памяти
команд в сумматоре.
*
```

```
TABLEW LAR AR4,#300h; AR4 содержит источник адресов в памяти данных
MAR *,AR4           ; ARP = AR4
RPT #511            ; Перемещает 512 значений из памяти данных в память команд
TBLW *+             ; Сумматор содержит dest. адреса память команд
RET
```

4.3.6.2.2 Перемещение данных из памяти программ в память данных

Команды BLPD и TBLR передают командные данные в память данных следующими способами:

- из внешней памяти программ во внешнюю память данных;
- из внешней памяти программ во внутреннюю память данных;
- из внутренней памяти программ во внутреннюю память данных;
- из внутренней памяти программ во внешнюю память данных.

Когда никакая внешняя память недоступна, память программ может содержать необходимую таблицу коэффициентов, которые должны быть загружены во внутреннюю память данных. Пример показывает, как использовать команду BLDP при перемещении внешней памяти программ во внутренний DARAM блок B1.

Также можно использовать команду TBLR при передаче командных данных в память данных. Команда TBLR отличается от команды BLPD, в которой сумматор содержит источник адресов памяти программ. Это дает определенные особенности расчета, которые проходят быстрее, чем предыдущие, а также в размещении блока данных в памяти программ. Следующие примеры показывают, как использовать команду TBLR при перемещении внешних команд во внутренний DARAM блок B1.

```
*
* Это стандартный пример использования команды BLPD при перемещении внешней
памяти программ во внутреннюю память данных. Он также может использоваться для
загрузки таблицы коэффициентов, сохраняющихся во внешней памяти программ в
памяти данных, когда никакая внешняя память данных недоступна.
*
MOVEPD LAR AR7,#300h ; AR7 содержит dest. адреса в памяти данных
MAR *,AR7            ; ARP=AR7
RPT #127           ; Перемещает 128 значений из памяти команд в блок данных B1
BLPD #0FD00h,*+
RET
```

Перемещение внешних команд во внутреннюю память данных с помощью команды TBLR:

```
*
* Это стандартный пример использования команды TBLR при перемещении внешней памяти программ во внутреннюю память данных. Эта процедура должна содержать источник адресов памяти команд в сумматоре.
*
TABLER LAR AR3,#300h ; AR3 содержит dest. адреса в памяти данных
MAR *,AR3             ; ARP=AR3
RPT #127           ; Перемещает 128 значений из памяти команд в блок данных B1
TBLR *+            ; Сумматор содержит адреса внешней памяти команд
RET
```

4.3.6.2.3 Перемещение данных из памяти

Команда LMMR может использоваться при передаче данных из внешней или внутренней памяти данных во внешний порт ВВОДА/ВЫВОДА. Пример показывает, как использовать команду LMMR при перемещении данных из внутренней памяти данных в порт ввода/вывода картированной памяти.

Перемещение данных из внутренней памяти данных в пространство ввода/вывода при помощи команды LMMR:

```
*
* Это стандартный пример использования команды LMMR при перемещении данных из внутренней памяти данных в порт I/O отображаемый в памяти. Обратить внимание, что 16 портов I/O отображаемых в 0 странице данных карты памяти 1867ВЦ2Т, 1867ВЦ2АТ.
*
OUTPUT:
LDP #0   ; DP=0
RPT #63  ; Перемещает 64 значения из таблицы, начинающейся с 800h в данные
LMMR 50h,#800h ; Память у порта 50h.
RET
```

4.3.6.2.4 Перемещение данных из пространства ввода/вывода в память данных при помощи команды SMRR

Команда SMRR может использоваться при передаче данных из внешнего порта ввода/вывода во внутреннюю или внешнюю память данных. Пример показывает, как использовать команду SMRR при передаче данных из порта ввода/вывода отображаемой памяти во внутреннюю память данных.

```
*
* Это стандартный пример использования команды SMMR при перемещении данных из порта ввода/вывода отображаемой памяти. Примечательно то, что 16 портов ввода/вывода отображены в 0 странице данных карты памяти 1867ВЦ2Т, 1867ВЦ2АТ.
*
INPUT:
LDP #0 :DP=0
RPT #511 ; Перемещает 512 значений из порта 51h в таблицу, начинающуюся
SMMR 51h,#800h ; 800h в памяти данных.
RET
```

4.3.6.3 Внутрикристальное ПЗУ начальной загрузки

Главной функцией загрузчика операционной системы является передача кодов из внешнего источника в память программ при включении питания. Это может быть сделано несколькими различными способами, которые зависят от требований системы. В некоторых случаях последовательный интерфейс соответствует этим требованиям. Если коды уже сохранены в ПЗУ, параллельный интерфейс является наиболее подходящим.

Если вывод $MP/\overline{MC}$ процессора 1867ВЦ2Т (1867ВЦ2АТ) установлен низким во время аппаратного сброса, выполнение программы начинается с адреса 0000h внутренней ПЗУ. Этот адрес содержит команду перехода при начале выполнения программы загрузчика. Внутреннее ПЗУ заранее запрограммировано для выполнения данной операции. Программа загрузчика устанавливает регистры состояния центрального процессора перед инициализацией загрузчика операционной системы:

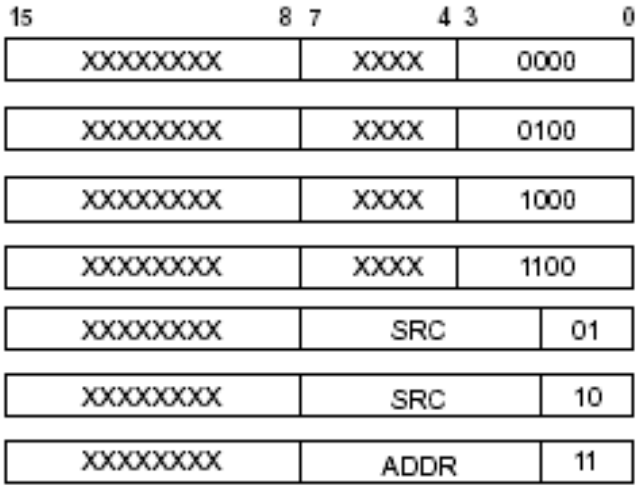
- прерывания глобально заблокированы (INTM=1);
- внутренний DARAM блок B0 отображен внутри пространства программ (CNF=1);
- внутренний блок SARAM отображен внутри пространства программ (RAM=1, OVLY=0);

– вся программа целиком и пространство памяти данных дают возможность работы с семью состояниями ожидания;

– 32К слов глобальной памяти данных разблокированы изначально в пространстве данных 8000h-FFFFh. После того, как передача кодов полностью завершена, глобальная память будет заблокирована перед тем, как управление будет передано по назначенному адресу.

Обратить внимание, что оба блока памяти (DARAM и SARAM) доступны в пространстве памяти программ; а это позволяет производить передачу кодов во внутреннюю память программ.

Программа загрузчика считывает глобальную память данных, расположенную в FFFFh по требованию шины (BR) и выводов стробированных данных (низкие) (DS). Нижние 8 битов слова с адресом FFFFh определяют режим загрузки, верхние 8 битов игнорируются загрузчиком.



X = Условия не соблюдаются

SRC = 6-разрядный адрес страницы для параллельного состояния EPROM

ADDR = 6-разрядный адрес страницы для режима перезаписи памяти

Рисунок 4.23 - Процедура загрузки выбранных слов для ПЗУ начальной загрузки

4.3.6.3.1 Режим последовательной загрузки

При выборе режима последовательной загрузки регистр SPC последовательного порта установлен в 00F8h для передачи 16-разрядного слова или в 00FCh для передачи 8-разрядного слова.

Внешний флаговый вывод сигнализирует, что процессор 1867ВЦ2Т (1867ВЦ2АТ) готов ответить. Вывод XF установлен высоким при сбросе и устанавливается низким при запуске приема. Никакие импульсы синхронизации кадра не могут появиться на FSR выводе, перед тем как XF станет низким.

4.3.6.3.2 Последовательная передача 16-разрядного слова

Если выбрана передача 16-разрядного слова (рисунок 4.24), первое 16-разрядное слово, полученное устройством из последовательного порта, устанавливает адресат кода в памяти команд. Следующее 16-разрядное слово указывает длину последующего кода. Эти два 16-разрядных слова сопровождаются N числом кодовых слов, которые будут переданы в память команд. Примечательно то, что число 16-разрядных слов, указанных параметрами N, не включает в себя первые два полученных 16-разрядных слова. После указанное число кодовых слов передается в память команд, и процессор 1867ВЦ2Т (1867ВЦ2АТ) совершает переход по указанному адресу. Длина N определяется как:

Длина N = число 16-разрядных слов – 1

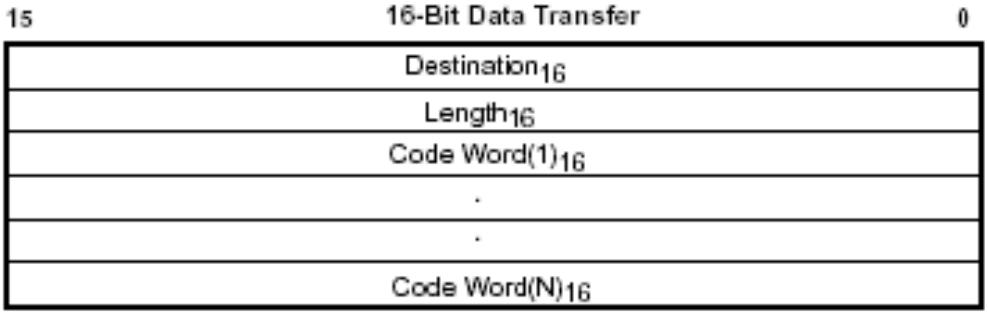


Рисунок 4.24 - Передача 16-разрядных слов для ПЗУ начальной загрузки

Условные обозначения: $Distination_{16}$ - 16-разрядный указанный адрес;
$Lenght_{16}$ - 16-разрядное слово, которое указывает длину кода (N), который следует;
$Code\ Word_{16}$ – N число 16-разядных слов при передаче.

4.3.6.3.3 Последовательная передача 8-разрядного слова

Если выбрана передача 8-разрядного слова (рисунок 4.25), байт более высокого порядка и байт более низкого порядка формируют 8-разрядное слово. Первое 16-разрядное слово, полученное устройством из последовательного порта, указывает необходимые

адреса кодов в памяти команд. Следующее 16-разрядное слово указывает длину фактического следующего кода. Эти два 16-разрядных слова сопровождаются за N числом кодовых слов, которые будут переданы в память команд. Обратить внимание, что число 16-разрядных слов указанных параметром N, не включает в себя первые два 16-разрядных слова. После указанное число кодовых слов передается в память команд, процессор выполняет переход по необходимому адресу. Длина N определяется как:

Длина N = число 16-разрядных слов – 1

или

Длина N =(число байтов, которые будут переданы,/2) – 1.

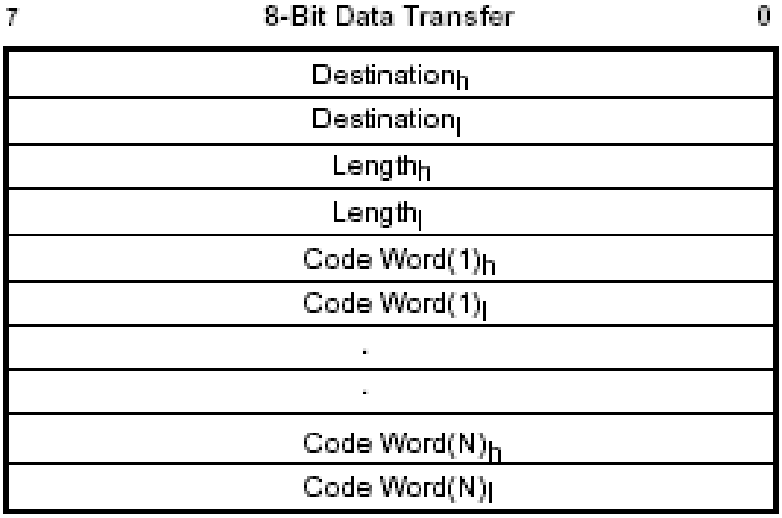


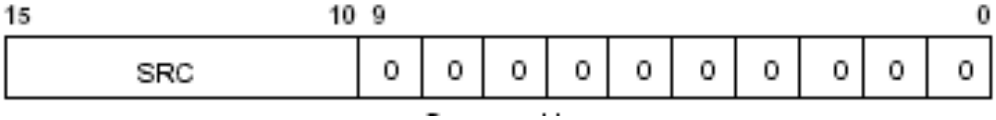
Рисунок 4.25 - Передача 8-разрядного слова для ПЗУ начальной загрузки

Условные обозначения: Destination$_h$ – Старший байт указанного адреса;
Destination$_l$ - Младший байт указанной длины;
Length$_h$ – Старший байт, указывающий длину следующего кода;
Length$_l$ - Младший байт, указывающий длину следующего кода;
Code Word (N)$_h$ – Старший байт N числа 16-разрядных слов, которые будут переданы;
Code Word (N)$_l$ – Младший байт N числа 16-разрядных слов, которые будут переданы.

4.3.6.3.4 Режим параллельной загрузки программируемого ПЗУ (EPROM)

Режим параллельной загрузки программируемого ПЗУ (EPROM) используется только в случае, если коды сохранены в EPROM (8-разрядная и 16-разрядная ширина). Коды передаются из глобальной памяти данных (начиная с указанного адреса). Шесть

самых старших двоичных разрядов указанного адреса (MSBs) указаны SRC полем выбранного слова процедуры загрузки. 16-разрядный исходный адрес определяется SRC полем, как это показано на рисунке 4.26. Процессор 1867ВЦ2Т (1867ВЦ2АТ) передает контроль исходному адресу после блокировки глобальной памяти данных.



Исходные адреса

Рисунок 4.26 - 16-разрядный исходный адрес для режима параллельной загрузки EPROM

Условные обозначения: SRC – 16-разрядная страница адресов.

4.3.6.3.5 Параллельная передача 16-разрядного слова

Если выбран режим параллельной загрузки 16-разрядного слова, код загрузки будет считываться в 16-разрядном слове, начиная с исходного адреса. Исходный адрес увеличивается на 1 после каждой операции чтения. Первое 16-разрядное слово, считанное из исходного адреса, указывает адрес назначения кода в памяти команд. Следующее 16-разрядное слово указывает длину фактического кода, который следует далее. Эти два 16-разрядных слова сопровождаются N числом кодовых слов, которые будут переданы памяти команд. Примечательно то, что число 16-разрядных слов, указанных параметрами N, не включает в себя первые два полученных 16-разрядных слова. После указанное число кодовых слов передается памяти команд, процессор 1867ВЦ2Т (1867ВЦ2АТ) передает контроль исходному адресу. Длина N определяется как:

Длина N = число 16-разрядных слов – 1.

Обратить внимание на то, что существует задержка не менее чем в 4 командных цикла, между чтением из EPROM и записью в указанный адрес. Эта задержка гарантирует, что если адресат находится во внешней памяти (например, быстрый SRAM), то будет достаточно времени для выключения исходной памяти (например, EPROM) перед началом выполнения операции записи.

4.3.6.3.6 Параллельная передача 8-разрядного слова

Если выбран режим параллельной загрузки 8-разрядного слова (рисунок 4.25), два последовательных расположения памяти (начиная с исходного адреса) читаются при

создании 16-разрядного слова. Наиболее высокий байт 16-разрядного слова сопровождается наиболее низким байтом. Данные считываются из самых младших восьми строк линии данных, при этом игнорируется самый старший байт шины данных. Первое 16-разрядое слово указывает адресат кода в памяти команд. Следующее 16-разрядное слово указывает длину фактического следующего кода. Эти два 16-разрядных слова следуют за N числом кодовых слов при передаче в память команд. Обратить внимание на то, что количество 16-разрядных слов, указанное параметром N, не включает в себя первые 4 полученных байта (первые два 16-разрядных слова). После указанное число кодовых слов передается в память команд, процессор 1867ВЦ2Т (1867ВЦ2АТ) передает контроль исходному адресу. Длина N определяется как:

Длина N = число 16-разрядных слов – 1

или

Длина N =(число байтов, которые будут переданы,/2) – 1.

Обратите внимание на то, что существует задержка продолжительностью не менее 4 командных циклов, между чтением из EPROM и записью по указанному адресу. Эта задержка гарантирует, что если адресат находится во внешней памяти (например, быстрый SRAM), то будет достаточно времени для выключения исходной памяти (например, EPROM) перед началом выполнения операции записи.

4.3.6.3.7 Режим параллельной загрузки ввода/вывода

Режим параллельной загрузки ВВОДА/ВЫВОДА асинхронно передает коды из порта ввода/вывода к адресу 50h внутренней или внешней памяти. Каждое слово может содержать 16 или 8 битов. Процессор 1867ВЦ2Т (1867ВЦ2АТ) соединяется с внешними устройствами через BIO или XF линии. Это облегчает соединение внешнего более медленного процессора с процессором 1867ВЦ2Т (1867ВЦ2АТ) через запрашивание/управление (установление) линий BIO и XF. Протокол установления связи, показанный на рисунке 4.27, должен использоваться при успешном переносе через порт ввода/вывода 50h.

Если выбран 8-разрядный режим загрузки, два последовательных 8-разрядных слова считываются при формировании 16-разрядного слова. Самый старший байт 16-разрядного слова сопровождается самым младшим байтом. Данные считываются из

самых младших линий данных ввода/вывода порта 50h, игнорируя старшие байты на шине данных.

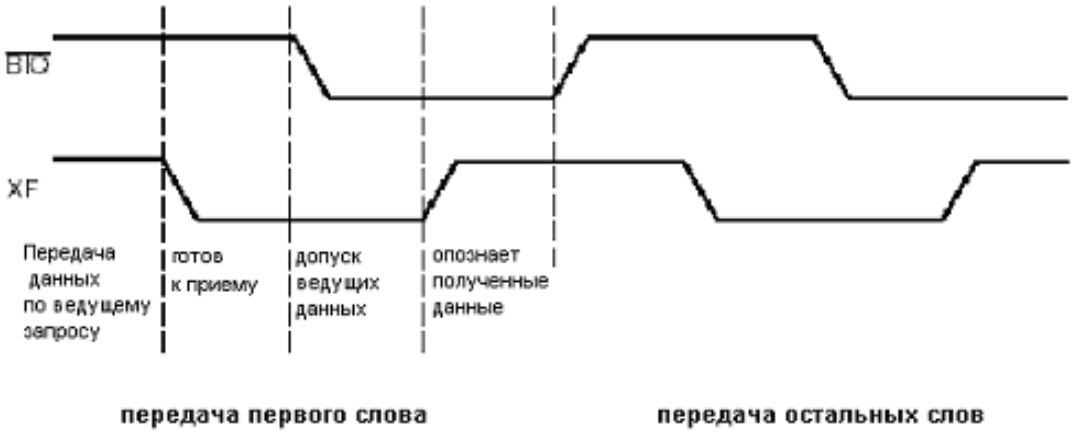


Рисунок 4.27 - Протокол установления связи

Обратить внимание на то, что существует задержка длиной не менее 4 командных циклов между передним фронтом XF и операцией записи по указанному адресу. Эта задержка обеспечивает то, что если адресат расположен во внешней памяти (например, быстрый SRAM), то ведущий процессор располагает достаточным временем для выключения буферов данных перед тем, как начнется выполнение операции записи. Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет доступ к внешней шине только тогда, когда высокий XF.

4.3.6.3.8 Режим перезапуска из памяти

В режиме перезапуска загрузчик запускается, но не перемещает никакие коды. Контроль передается адресу входа. Режим перезапуска из памяти может использоваться в том случае, если команда уже передана внутренней или внешней памяти другими способами (например, HPI или внешний прямой доступ) или если требуется сброс перезапускающего устройства. Шесть MSBs адреса входа указываются полем ADDR выбранного слова режима перезагрузки. 16-разрядный адрес входа определяется через поле ADDR, как это показано на рисунке 4.28. Процессор 1867ВЦ2Т (1867ВЦ2АТ) передает контроль внешнему адресу после блокировки глобальной памяти данных.

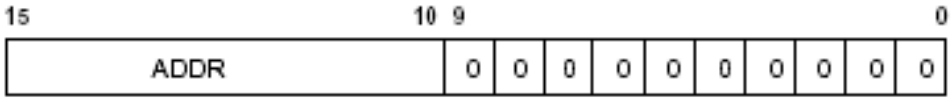


Рисунок 4.28 - 16-разрядный адрес входа для режима перезапуска из памяти

Условные обозначения: ADDR – 6-разрядная страница адресов.

4.3.6.3.9 Внешняя параллельная операция интерфейса

Все шинные циклы включают целое число циклов CLKOUT1. Один цикл CLKOUT1 определяется как расстояние от заднего фронта CLKOUT1 до следующего заднего фронта CLKOUT1. Запись непосредственно предшествует чтению или непосредственно следует за требованием чтения три цикла. Обратимся к рисункам 4.29, 4.30 и 4.31 для синхронизации двух циклов (чтения и записи).

Для цикла чтения $\overline{\text{STRB}}$ становится низким, и ADDRESS становится допустимым на заднем фронте CLKOUT1. Для цикла чтения с нулевым состоянием ожидания RD сигнал становится низким на переднем фронте CLKOUT1 и потом становится высоким при следующем заднем фронте CLKOUT1. Для цикла чтения с единичным состоянием ожидания (мультицикл) RD остается низким, но становится высоким на заднем фронте CLKOUT1 перед следующим циклом, даже если циклы непрерывны. Чтение данных выбирается на преднем фронте $\overline{\text{R}}/\overline{\text{D}}$.

$\text{R}/\overline{\text{W}}$ сигнал становится высоким не менее чем на половину цикла CLKOUT1 перед некоторым циклом чтения; для непрерывного цикла чтения $\overline{\text{STRB}}$ становится низким. В конце цикла чтения или при последовательности операций чтения $\overline{\text{STRB}}$ или $\overline{\text{R}}/\overline{\text{D}}$ становятся высокими на заднем фронте CLKOUT1.

Цикл записи всегда имеет не менее чем один неактивный цикл CLKOUT1 перед и после действительной операции записи, включая продолжительную запись. Это позволяет совершать плавный переход между операциями записи и любыми смежными шинными операциями или операциями записи. Для этого цикла $\overline{\text{STRB}}$ и $\overline{\text{W}}/\overline{\text{E}}$ всегда высокие. Сигнал $\text{R}/\overline{\text{W}}$ всегда изменяет состояние на переднем фронте CLKOUT1 во время цикла перед и после записи или серии операций записи. Это предотвращает конфликтную ситуацию при обращении к шине во время перехода между операциями чтения и записи. Обратить внимание на то, что во время серии операций записи $\text{R}/\overline{\text{W}}$ остается низким.

Синхронизация действительных адресов при записи отличается в зависимости от того, какие действия имели место перед и после операции записи. Между операциями записи и для первой и последней операции записи в серии ADDRESS становится доступен для цикла чтения (половина цикла CLKOUT1 ранее) на переднем фронте быстрее

CLKOUT1, чем на заднем фронте CLKOUT1. Это исключение при обычной синхронизации адресов цикла чтения.

Для действительной операции записи $\overline{\text{STRB}}$ и $\overline{\text{W}}/\overline{\text{E}}$ становятся низкими на заднем фронте CLKOUT1 и становятся низкими впредь до следующего заднего фронта CLKOUT1 (для цикла записи с нулевым состоянием ожидания). Для записи с единичным состоянием ожидания (мультицикл) $\overline{\text{STRB}}$ и $\overline{\text{W}}/\overline{\text{E}}$ остаются низкими, но снова становятся высокими на заднем фронте CLKOUT1 при начале цикла. Запись данных управляется приблизительно у заднего фронта $\overline{\text{STRB}}$ и $\overline{\text{W}}/\overline{\text{E}}$ и удерживается приблизительно для одной половины цикла CLKOUT1 после того, как $\overline{\text{STRB}}$ и $\overline{\text{W}}/\overline{\text{E}}$ стали высокими.

Переходы на выводы управления внешним параллельным интерфейсом ($\overline{\text{STRB}}$, CLKOUT1, $\overline{\text{W}}/\overline{\text{E}}$, $\overline{\text{R}}/\overline{\text{D}}$) вызываются теми же самыми внутренними тактовыми сигналами. Пока эти сигналы используют ту же самую схему буферов выхода, они все переключаются внутри своих закрытых доступов, как указывается в листе данных для процессора 1867ВЦ2Т (1867ВЦ2АТ).

Переходы на адресные шины и другие имеющие отношение выходы ($\overline{\text{IS}}$, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\text{R}/\overline{\text{W}}$, BR) вызываются теми же внутренними сигналами, которые обуславливают переходы на выходы управления; кроме того, внутренняя логика устройства, которое генерирует эти выходы, отличается от цепи, используемой в контрольных выходах. Переходы на адресные шины и другие имеющие отношение выходы обычно имеют место позже, чем переходы на контрольных линиях.

Синхронизация выходов управления относительно CLKOUT1 управляется в листе данных процессора 1867ВЦ2Т (1867ВЦ2АТ). Синхронизация адресов относительно CLKOUT1 может получаться из синхронизации адресов для сигналов управления и сигналов управления синхронизацией для CLKOUT1. Например, задержка сигнала CLKOUT1 на заднем фронте к соответствующей адресной шине, к началу цикла чтения определяется как:

H - (установка адреса в RD)) + максимальный положительный RD к скосу CLKOUT1 (относительно листа данных 1867ВЦ2Т, 1867ВЦ2АТ для определенных значений синхронизации).

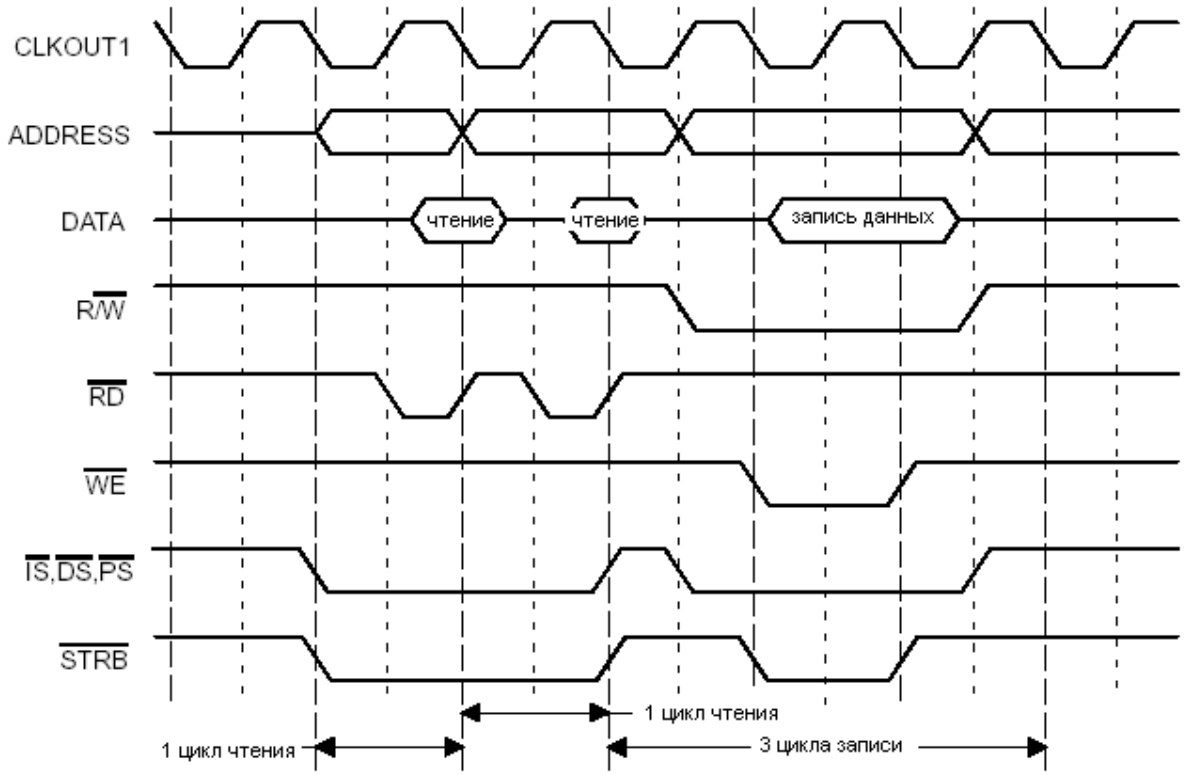


Рисунок 4.29 - Внешние операции интерфейса для чтения-чтения-записи (нулевое состояние ожидания)

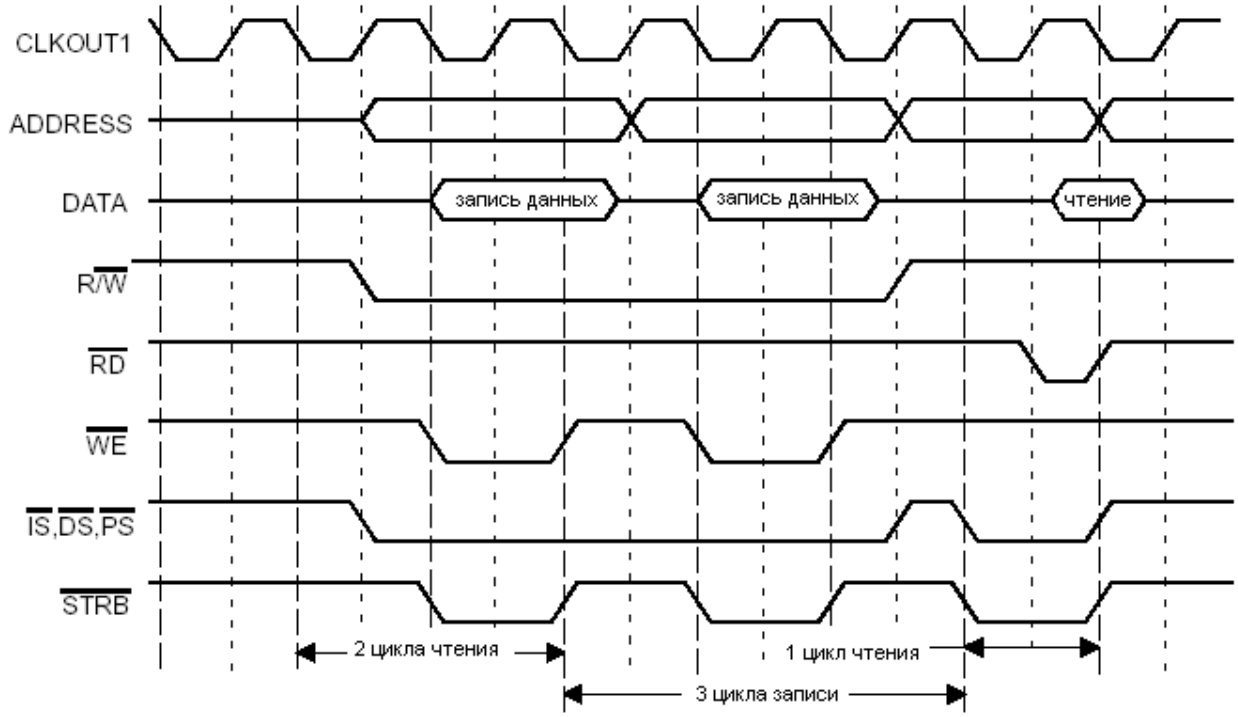


Рисунок 4.30 - Внешние операции интерфейса для записи-записи-чтения (нулевое состояние ожидания)

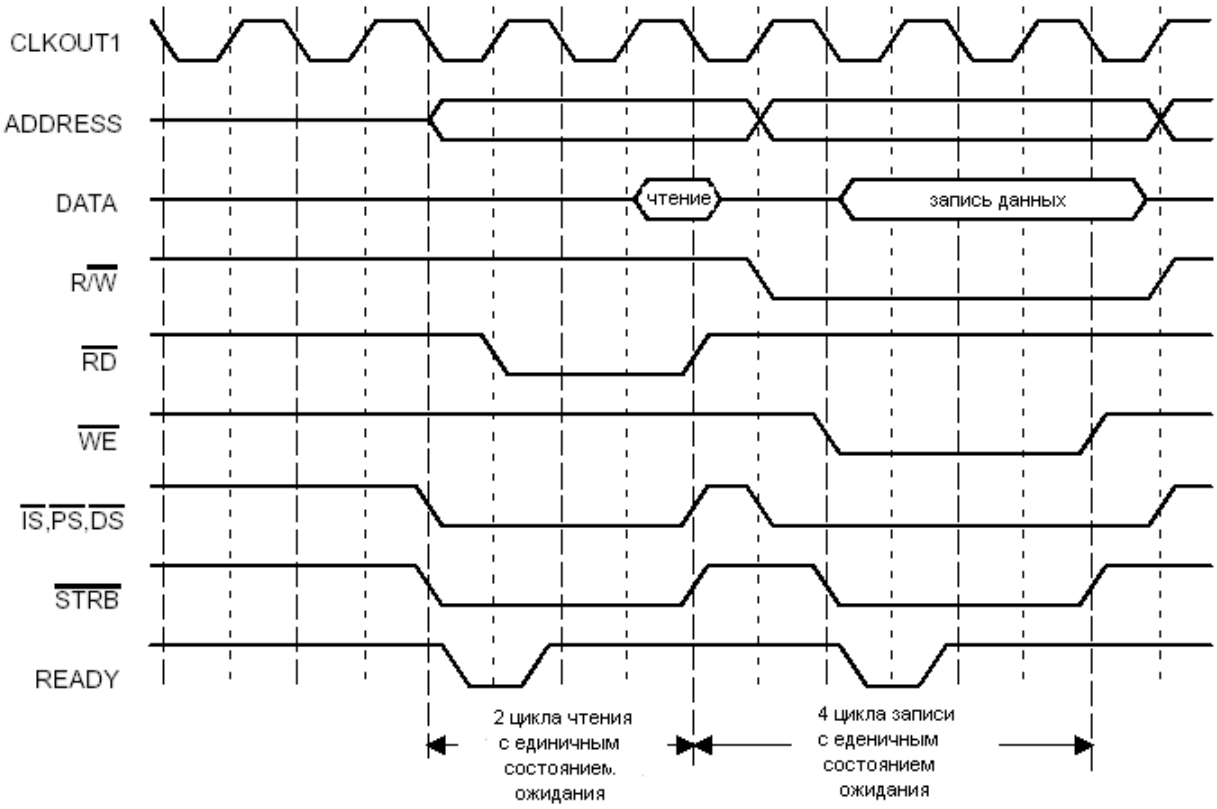


Рисунок 4.31 - Внешние операции интерфейса для чтения-записи (единичное состояние ожидания)

4.3.6.4 Программные генераторы состояния ожидания

Программные генераторы состояния ожидания могут использоваться, чтобы удлинять внешние шинные циклы до семи машинных циклов. Все внешние считывания требуют не меньше одного машинного цикла, в то время как все внешние записи требуют не менее двух машинных циклов. Кроме того, внешняя запись, непосредственно следующая или непосредственно предшествующая внешнему циклу чтения, требует не менее трех машинных циклов. Это обеспечивает удобный внешний доступ внешнего интерфейсного устройства, которое не удовлетворяет нас по времени доступа к процессору 1867ВЦ2Т (1867ВЦ2АТ). Процессор 1867ВЦ2Т (1867ВЦ2АТ) может генерировать состояния при расширении циклов чтения/записи памяти программируемым генератором состояния ожидания или интерфейсом с аппаратной строкой READY. Программируемые генераторы состояния ожидания могут генерировать только до семи состояний ожидания. Внешнее устройство, требующее более чем семь состояний ожидания, может использовать аппаратную строку READY при создании состояния ожидания.

Таблица 4.17 показывает количество циклов, необходимых для различных типов доступов внешних устройств.

Таблица 4.17 - Число циклов CLKOUT1 доступа для различного числа состояний ожидания

Номер состояния ожидания	Номер цикла CLKOUT1			
	Системное ожидание		Программное ожидание	
	чтение	запись	чтение	запись
0	1	2n+1	1	2n+1
1	2	3n+1	2	3n+1
2	3	4n+1	3	4n+1
3	4	5n+1	4	5n+1

Также обратите внимание на то, что внешний вход READY выбирается только после того, как внутреннее программное состояние ожидания завершено. Кроме того, если вход READY устанавливается низким перед завершением внутреннего программного состояния ожидания, то состояния с нулевым ожиданием не добавляются к внешнему доступу памяти пока указанное число программных состояний ожидания не завершено. Состояния ожидания добавляются только, если вход READY все еще низкий после завершения программного состояния ожидания. Можно еще добавить то, что вход READY не асинхронный вход и не вход установления, и время удержания этого сигнала, как указано в листе данных для процессора 1867ВЦ2Т (1867ВЦ2АТ), должно быть выполнено.

4.4 Периферийные устройства

Вутрикристальные периферийные устройства, соединенные с ядром процессора 1867ВЦ2Т (1867ВЦ2АТ) включают в себя: ФАПЧ (внешняя частота x1), таймер, программируемые генераторы состояния ожидания, внешние входы-выходы общего назначения, порты ввода/вывода, последовательные порты.

Периферийные устройства управляются через картированные в памяти регистры. Последовательные порты и таймер синхронизируются с процессором при помощи прерываний. Периферийные устройства и управление периферийными устройствами представлены в этой главе, как показано ниже.

4.4.1 Управление периферийными устройствами

Работа и управление периферийных сетей осуществляется через обращение к картированным в памяти регистрам данных и управления. Работа последовательных портов и таймера синхронизируется с процессором через прерывания или посредством

опроса соответствующих флагов. Устанавливая или очищая разряды, периферийные устройства могут быть включены, выключены, инициализированы и динамически переконфигурированы. Данные передаются в и из периферийных устройств через картированные в памяти регистры данных. Когда периферия не используется, внутренняя синхронизация может быть отключена от периферии, позволяя снизить энергопотребление в нормальном режиме работы или в режиме IDLE.

4.4.1.1 Картированные в памяти регистры и порты ввода/вывода

Имеются 28 регистров процессора, картированных в пространстве памяти данных. В дополнение к этим регистрам оперативной памяти имеются 17 регистров периферийных устройств и 16 портов ввода/вывода, адресуемых в пространстве памяти данных. В таблице 4.18 перечислены отображаемые в памяти регистры процессора 1867ВЦ2Т (1867ВЦ2АТ). Все записи в эти регистры (к ним не относятся регистры процессора и порты ввода/вывода) требуют одного дополнительного цикла CLKOUT1.

Таблица 4.18 - Картированные в памяти регистры процессора

Картированные в памяти регистры процессора			
Имя	Адрес		Описание
	Dec	Hex	
1	2	3	4
---	0-3	0-3	Зарезервировано
IMR	4	4	Регистр маски прерывания
GREG	5	5	Регистр распределения глобальной памяти
IFR	6	6	Регистр флага прерывания
PMST	7	7	Регистр состояния режима процессора
RPTC	8	8	Регистр счетчика повторений
BRCR	9	9	Регистр счетчика повторений блока
PASR	10	A	Регистр стартового адреса программы повторений блока
PAER	11	B	Регистр конечного адреса программы повторений блока
TREG0	12	C	Временный регистр, используемый для сомножителей
TREG1	13	D	Временный регистр, используемый для счета динамического сдвига
TREG2	14	E	Временный регистр, используемый как указатель бита при тестировании динамического бита
DBMR	15	F	Регистр управления динамическим битом
AR0	16	10	Вспомогательный регистр ноль
AR1	17	11	Вспомогательный регистр один
AR2	18	12	Вспомогательный регистр два
AR3	19	13	Вспомогательный регистр три
AR4	20	14	Вспомогательный регистр четыре
AR5	21	15	Вспомогательный регистр пять
AR6	22	16	Вспомогательный регистр шесть

Продолжение таблицы 4.18

1	2	3	4
AR7	23	17	Вспомогательный регистр семь
INDX	24	18	Индексный регистр
ARCR	25	19	Регистр сравнения вспомогательный регистров
CBSR1	26	1A	Начальный регистр циклического буфера 1
CBER1	27	1B	Конечный регистр циклического буфера 1
CBSR2	28	1C	Начальный регистр циклического буфера 2
CBER2	29	1D	Конечный регистр циклического буфера 2
CBCR	30	1E	Регистр управления циклическим буфером
BMAR	31	1F	Регистр адреса перемещения блока
Картированные в памяти регистры периферийных устройств			
DRR	32	20	Регистр приема данных
DXR	33	21	Регистр передачи данных
SPC	34	22	Регистр управления последовательным портом
---	35	23	Зарезервировано
TIM	36	24	Регистр таймера
PRD	37	25	Регистр периода
TCR	38	26	Регистр управления таймером
---	39	27	Зарезервировано
PDWSR	40	28	Регистр состояния ожидания программы/данных
IOWSR	41	29	Регистр состояния ожидания ввода/вывода
CWSR	42	2A	Регистр управления состоянием ожидания
---	43-47	2B-2F	Зарезервировано
TRCV	48	30	Приемный регистр данных TDM
TDXR	49	31	Передающий регистр данных TDM
TSPC	50	32	Регистр управления последовательным портом TDM
TCSR	51	33	Регистр выбора канала TDM порта
TRTA	52	34	Регистр адреса приема/передачи TDM порта
TRAD	53	35	Регистр принятого адреса TDM порта
Отображаемые в памяти порты ввода/вывода TDM			
---	54-79	36-4F	Зарезервировано
PA0	80	50	Порт 0 ввода/вывода
PA1	81	51	Порт 1 ввода/вывода
PA2	82	52	Порт 2 ввода/вывода
PA3	83	53	Порт 3 ввода/вывода
PA4	84	54	Порт 4 ввода/вывода
PA5	85	55	Порт 5 ввода/вывода
PA6	86	56	Порт 6 ввода/вывода
PA7	87	57	Порт 7 ввода/вывода
PA8	88	58	Порт 8 ввода/вывода
PA9	89	59	Порт 9 ввода/вывода
PA10	90	5A	Порт 10 ввода/вывода
PA11	91	5B	Порт 11 ввода/вывода
PA12	92	5C	Порт 12 ввода/вывода
PA13	93	5D	Порт 13 ввода/вывода
PA14	94	5E	Порт 14 ввода/вывода
PA15	95	5F	Порт 15 ввода/вывода

4.4.1.2 Внешние прерывания

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет четыре внешних маскируемых пользователем прерывания ($\overline{INT4} - \overline{INT1}$), доступные внешним устройствам для прерывания процессора, и одно немаскируемое прерывание ($\overline{NMI}$). Внутренние прерывания генерируются таймером (TINT), последовательным портом (RINT и XINT), портом TDM (TRNT и TXNT) и инструкциями программного прерывания (TRAP, NMI и INTR). Приорететы прерыванийустановлены так, что сброс ($\overline{RS}$), имеет наибольший приоритет, а $\overline{INT4} - \overline{INT1}$ имеет наименьший приоритет. $\overline{NMI}$ имеет второй приоритет после $\overline{RS}$.

Этот подраздел подробно описывает организацию и управление прерывания. Ячейки и приоритеты для всех внутренних и внешних прерываний показаны в таблице 4.19. Инструкция TRAP, используемая для программного прерывания, не приоритезируется, но включена сюда, поскольку имеет собственную векторную ячейку.

Расположение векторов прерывания в ячейках, определенных пятиразрядным полем IPTR регистра PMST, и значения адресов показаны в таблице 4.19. Поле IPTR устанавливается в нуль при сбросе устройства, кончающееся векторами, начинающимися с 0000h в пространстве памяти программ. Векторный адрес программы может быть переотображен в начало любого из 32 блоков, состоящих из 2К слов, содержащего пространство памяти программ. Это делается посредством загрузки пятиразрядного адреса блока (5 младших разрядов полного 16-разрядного адреса) в IPTR. Например, векторы могут быть передвинуты к началу внутрикристального ОЗУ программы процессора 1867ВЦ2Т (1867ВЦ2АТ) посредством загрузки IPTR единицей. Когда встречается внутреннее прерывание, значение в IPTR загружается в пять старших разрядов адреса вектора, а условный адрес прерывания, вызывающий внутреннее прерывание, составляет 6 младших разрядов адреса вектора. Эта схема условной адресации применима и ко всем десяти прерываниям и к программному внутреннему прерыванию. Она не применяется к вектору сброса, поскольку сигнал сброса вызывает IPTR, чтобы установить в нуль.

Таблица 4.19 - Ячейки и приоритеты прерывания

Имя	Ячейка		Приоритет	Функция
	Dec	Hex		
$\overline{RS}$	0	0	1 (наибольший)	Сигнал внешнего сброса
$\overline{NMI}$	36	24	2	Немаскируемое прерывание
$\overline{INT1}$	2	2	3	Внешнее пользовательское прерывание #1
$\overline{INT2}$	4	4	4	Внешнее пользовательское прерывание #2
$\overline{INT3}$	6	6	5	Внешнее пользовательское прерывание #3
TINT	8	8	6	Прерывание внутреннего таймера
RINT	10	A	7	Прерывание приема последовательного порта
XINT	12	C	8	Прерывание передачи последовательного порта
TRNT	14	E	9	Прерывание приема TDM порта
TXNT	16	10	10	Прерывание передачи TDM порта
$\overline{INT4}$	18	12	11	Внешнее пользовательское прерывание #4
---	20-33	14-21	N/A	Зарезервировано
TRAP	34	22	N/A	Вектор инструкции внутреннего прерывания
---	38-41	26-29	N/A	Зарезервировано

Когда происходит прерывание, оно хранится в 16-разрядном регистре флага прерывания (IFR). Каждое прерывание хранится в IFR до тех пор, пока оно будет принято. Флаг может быть очищен любым из трёх следующих способов:

1) сброс устройства ($\overline{RS}$ активный низкий), или
2) программа принимает внутреннее прерывание, или
3) программа записывает единицу в соответствующий бит в IFR.

IFR размещается по адресу 06h в пространстве памяти данных и может считываться для идентификации активного прерывания и записываться для очистки прерываний. Регистр IFR располагается следующим образом:

15-9	8	7	6	5	4	3	2	1	0
$\overline{RESET}$	$\overline{INT4}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{INT3}$	$\overline{INT2}$	$\overline{INT1}$

Единица в определенном бите, при чтении, указывает на активное прерывание. Например, если IFR читается как 0005h, то активны $\overline{INT3}$ и $\overline{INT1}$. Единица может записываться в определенный бит для очистки определенного прерывания. В данном примере, если единица записывается в нулевой бит (0001h в IFR), то прерывание $\overline{INT1}$ будет очищено. Все активные прерывания могут быть очищены посредством считывания IFR с последующей записью значения обратно в IFR. В вышеописанном примере значение

0005h должно быть записано обратно в IFR для очистки обоих ожидающих решения прерываний.

Когда принимается внутреннее прерывание, автоматически очищается соответствующий флаг прерывания. Когда ЦПУ принимает прерывание и выбирает программно-управляемой вектор, оно генерирует сигнал подтверждения приема прерывания ($\overline{\text{IACK}}$), который очищает соответствующий флаговый разряд прерывания. Аппаратный сброс ($\overline{\text{RS}}$ активный низкий) очищает все ожидающие решения флаги прерывания.

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет отображаемые в памяти регистры маски прерывания (IMR) для маскирующих внешних и внутренних прерываний. Располагается регистр следующим образом:

15-9	8	7	6	5	4	3	2	1	0
$\overline{\text{RESET}}$	$\overline{\text{INT4}}$	TXNT	TRNT	XINT	RINT	TINT	$\overline{\text{INT3}}$	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$

Единица в позициях 7 - 0 регистра IMR включает соответствующее прерывание, обуславливая таким образом INTM = 0. IMR доступен как для операций чтения, так и записи. Когда читается IMR, неиспользуемые биты (с 15 по 8) читаются как один. Младшие девять разрядов используются для записи или чтения из IMR. Обратите внимание, что $\overline{RS}$ не включен в IMR.

Прерывания могут быть асинхронно зафиксированны. В функциональной логической организации для $\overline{\text{INT}(4-1)}$, показанной на рисунке 4.32, внешнее прерывание $\overline{\text{INTn}}$ синхронизируется с ядром через пять синхронизирующих триггеров-защёлок. Фактическая реализация цепей прерываний совпадает с этим логическим представлением. Единица загружается в IFR, если обнаруживается последовательность 1-1-0-0-0 на пяти последовательных циклах CLKOUT1.

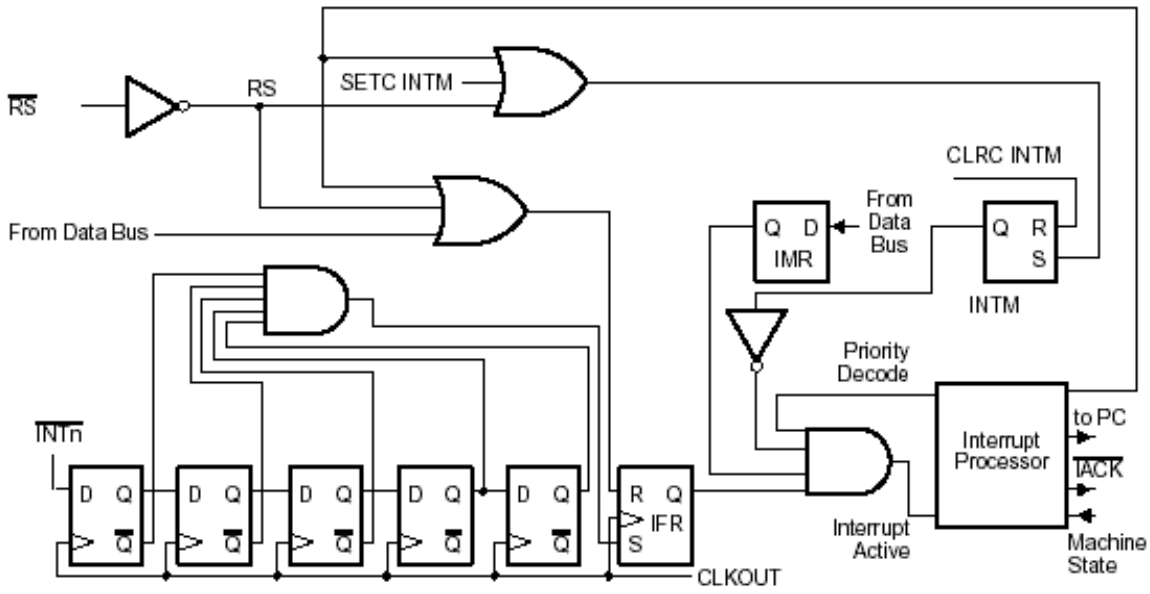


Рисунок 4.32 - Логическая диаграмма внешнего прерывания

Процессор 1867ВЦ2Т (1867ВЦ2АТ) многократно проверяет сигналы с контактов внешнего прерывания, чтобы избежать прерываний вызванных помехами. Чтобы обнаружить активное прерывание, эти устройства должны зафиксировать низкий уровень сигнала на протяжении трех последовательных машинных циклов. При обнаружении прерывания, устройство должно зафиксировать высокий уровень сигнала в двух последовательных машинных циклах, чтобы было возможно обнаружить следующее прерывание, контакты внешних прерываний проверяются по переднему фронту CLKOUT1. Если внешние прерывания работают асинхронно, то длительность импульсов должна быть увеличена чтобы гарантировать три низких уровня подряд. Заметим, что если процессор находится в режиме IDLE2, вход должен быть высоким минимум четыре такта CLKOUT1 и низким минимум пять машинных циклов, чтобы быть точно обнаруженным, когда инструкция INTR попадает в конвейер.

Если разряд INTM и регистр маски были выставлены правильно, процессор распознает сигнал прерывания. Когда инструкция INTR попадает в конвейер, устанавливается бит INTM и очищается соответствующий бит IFR, затем через три машинных цикла вырабатывается сигнал $\overline{\text{IACK}}$. $\overline{\text{NMI}}$ использует такую же логику, как $\overline{\text{INT}(4-1)}$, за исключением того, что не зависит от состояния регистра IMR (регистр маски прерывания) и состояния бита INTM.

4.4.1.3 Сброс периферийных устройств

Сброс определенным образом влияет на процессор 1867ВЦ2Т (1867ВЦ2АТ). В подразделе 4.1.8.1 описывается, что происходит в процессоре 1867ВЦ2Т (1867ВЦ2АТ),

когда активируется сброс. Что касается периферии - это сигнал $\overline{\text{SRESET}}$. Активизация сигнала $\overline{\text{SRESET}}$ приводит к следующим последствиям в системе периферийных устройств:

1) Оба регистра состояния ожидания (IOWSR, PDWSR) устанавливаются в 0FFFFh, делая максимальным время всех внешних обращений (7 машинных циклов). CWSR загружается с 0Fh.

2) Биты FO регистров SPC и TSPC устанавливаются в нуль, выбирая длину слова в 16 бит для каждого последовательного порта.

3) Биты FSM регистров SPC и TSPC устанавливаются в нуль. Для работы с синхроимпульсом FSR, FSM должен быть установлен в единицу.

4) Биты TXM регистров SPC и TSPC устанавливаются в нуль, задавая контакты FSX и TFSX как входы.

5) Биты SPC и TSPC загружаются с 0y00h, где 2 старших разряда Y составляют 10 (двоичную), а 2 младших разряда Y отражают текущий уровень на контактах приема и передачи тактов соответствующих портов.

6) Регистры TIM и PRD загружаются с 0FFFFh. Поле TDDR и бит TSS регистра TCR устанавливаются в нули и таймер стартует.

4.4.2 Тактовый генератор

Генератор внутренней частоты состоит из внутреннего осциллятора и цепи ФАПЧ (фазовой автоматической подстройки частоты), которые дают возможность выбрать источник синхронизации. Внутренний генератор управляется кварцевым резонатором или внешним источником частоты.

В таблице 4.20 показаны стандартные возможные режимы работы генератора.

Таблица 4.20 - Режимы работы генератора

CLKMD1	CLKMD2	Режим
0	0	Частота внешнего источника делится на 2, внутренний осциллятор отключен.
0	1	Зарезервировано для тестов
1	0	ФАПЧ (Частота внешнего источника x 1)
1	1	Частота внешнего источника / 2 или собственная частота с внешним кварцевым резонатором / 2

Когда выбрана функция «внутренняя частота / 2», то задействуется внутренний осциллятор путем подсоединения кварцевого резонатора к выводам X1 и X2/CLKIN.

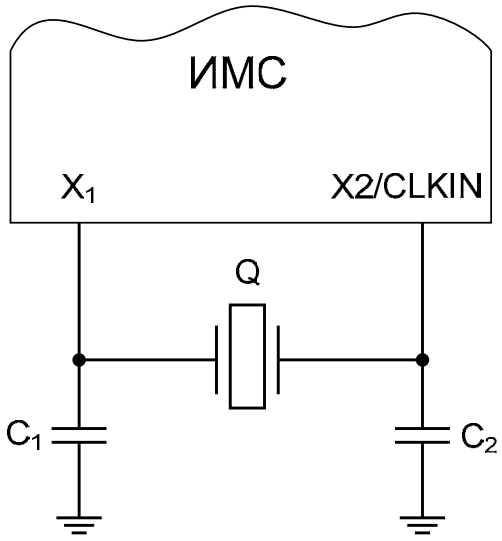
Частота CLKOUT1 в этом случае будет равна половине собственной частоты кварцевого резонатора.

Когда выбрана функция «внешняя частота / 2», то внешний источник частоты подсоединяется непосредственно к выводу X2/CLKIN, а X1 никуда не подсоединяется.

Внешняя частота делится пополам для выработки внутренней частоты.

Если выбирается функция «ФАПЧ», то внешний источник частоты подсоединяется непосредственно к выводу CLKIN2, X1 отключается от Ucc, X2/CLKIN подсоединяется к Ucc. Частота внешнего источника умножается на 1 и вырабатывается внутренняя частота.

Типовая схема включения микросхем с синхронизацией кварцевым резонатором до 20 МГц приведена на рисунке 4.33. Для работы ИМС на частотах свыше 20 МГц рекомендуется использовать кварцевый генератор, подключенный к входу внешней синхронизации X2/CLKIN.



Q - кварцевый резонатор до 20 МГц,

C_1, C_2 - подстроечные конденсаторы емкостью (4 ÷ 15) пФ

Рисунок 4.33 - Схема включения микросхем с синхронизацией кварцевым резонатором до 20 МГц

4.4.3 Таймер

Таймер – это внутрикристальный счетчик на декремент, который используется, чтобы периодически генерировать прерывания на центральном процессоре.

На рисунке 4.34 показана логическая блок-схема таймера.

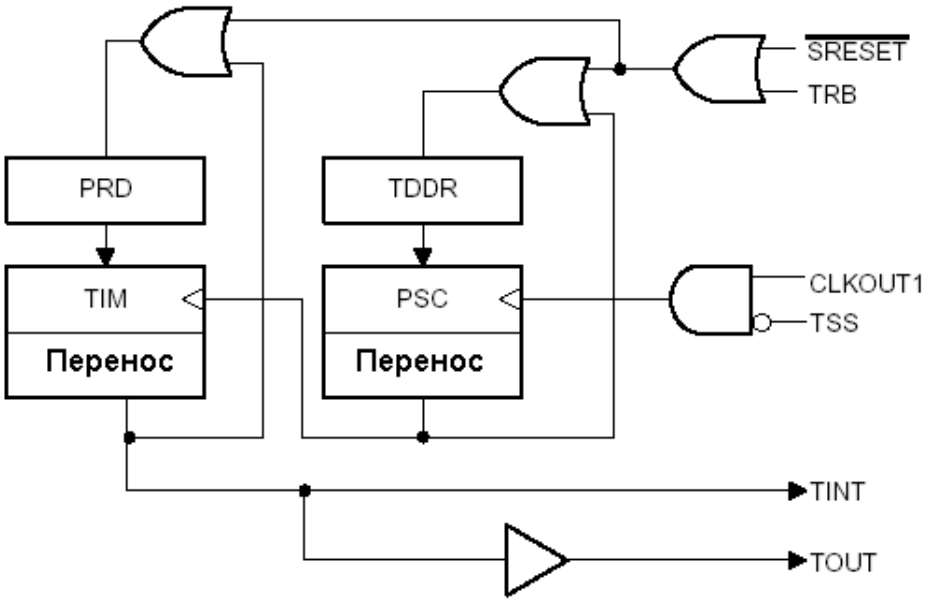


Рисунок 4.34 - Блок-схема таймера

Таймер управляется делителем, который уменьшается на 1 на каждом такте CLKOUT1. Прерывание от таймера (TINT) вырабатывается каждый раз, когда счетчик уменьшается до 0. Таймер обеспечивает удобный способдля периодичности ввода/вывода или может использоваться для других целей. Когда таймер остановлен (TSS=1), внутренние снхронизация таймера отключается, снижая энергопотребление..

4.4.3.1 Регистры таймера

Таймер управляется через регистр контроля (TCR), регистр счетчика таймера (TIM) и регистр периода таймера (PRD). Рисунок 4.35 и таблица 4.21 описывают разрядные поля регистра TCR.

15-12	11	10	9-6	5	4	3-0
Reserved	Soft	Free	PSC	TRB	TSS	TDDR

Рисунок 4.35 - Диаграмма контрольных регистров таймера

Таблица 4.21 - Биты контрольного регистра таймера

Разряд	Название	Значение сброса	Функция
1	2	3	4
15 – 12	Зарезерви-рованы		Это резервные биты и всегда они читаются как 0.

Продолжение таблицы 4.21

1	2	3	4
11	SOFT	0	Этот бит используется вместе с FREE битом, чтобы определить состояние таймера, когда происходит остановка. Когда FREE бит очищен, программный бит выбирает режим эмуляции. SOFT=0 Таймер останавливается немедленно, прерывая любую передачу. SOFT=1 Таймер останавливается после завершения передачи.
10	FREE	0–	Этот бит используется вместе с SOFT битом, чтобы определить состояние таймера, когда происходит остановка. Когда FREE бит очищен, программный бит выбирает режим эмуляции. FREE =0 Программный бит выбирает режим таймера. FREE =1 Таймер продолжает работать (независимо от SOFT бита).
9 – 6	PSC	–	Биты масштабирования для счета таймера. Эти биты определяют через сколько тактов CLKOUT1 таймер будет считать. Когда PSC уменьшается до нуля или таймер сбрасывается, PSC загружается содержимым TDDR и TIM уменьшается на 1.
5	TRB	–	Бит перезагрузки таймера. Этот бит сбрасывает внутрикристальный таймер. Когда TRB установлен, TIM загружается содержимым PRD и PSC загружается содержимым TDDR. TRB всегда читается как 0.
4	TSS	0	Статусный бит остановки таймера. Этот бит останавливает или запускает внутрикристальный таймер. При сбросе TSS бит очищается и таймер начинает считать. TSS=0 Таймер запускается. TSS=1 Таймер останавливается.
3 – 0	TDDR	0000	Биты этого регистра указывают таймеру коэффициент деления для поступающих тактовых импульсов. Когда PSC биты уменьшаются до нуля, PSC загружается содержимым TDDR.

4.4.3.2 Функционирование таймера

Когда PSC уменьшается до 0 или когда происходит сброс таймера путем установки бита TRB, содержимое TDDR загружается в PSC и TIM уменьшается на 1.

Когда TIM уменьшается до 0 или при сбросе таймера битом TRB, содержимое PRD загружается в TIM. Бит TRB читается как 0. Когда 1 записывается в TRB, таймер перезагружается, а TRB все равно читается как 0.

Замечание:

Текущее значение таймера может быть прочитано путем чтения TIM; значение PCS можно узнать прочитав TCR. Для того, чтобы прочитать оба этих значения, необходимо 2 инструкции. За это время счетчик может уменьшиться и мы можем прочитать неверные значения. Поэтому более корректно будет остановить таймер для чтения этих двух значений. Это можно сделать путем установки бита TSS и запустить снова очищением этого бита.

Частоту прерываний таймера (TINT) можно выразить формулой:

$$TINT_{частота} = \frac{1}{tc(C) \times U \times V} = \frac{1}{tc(C) \times (TDDR+1) \times (PRD+1)},$$

где $tc(C)$ - период CLKOUT1, U - сумма содержимого TDDR+1, V - сумма содержимого TDDR+1.

Частота прерываний TINT определяется частотой CLKOUT1 и двумя независимыми факторами. Два делителя представлены счетчиком на декремент и регистром периода на каждом этапе. Поля PSC и TDDR регистра TCR используются на первом этапе, а TIM и PRD на втором. Каждый раз, когда счетчик на декремент (PSC или TIM) уменьшается до нуля, вырабатывается перенос на следующем цикле CLKOUT1 и счетчик загружается из соответствующего регистра периода (TDDR или PRD). Выход второго этапа и есть сигнал TINT, который подается в ЦПУ и на выход таймера TOUT. Длительность импульса, который появляется на выходе второго этапа равна $t_c(C)$.

Таймер может быть использован для генерации частоты выборки для аналогового интерфейса. Следующий пример показывает использование таймера для гкнерации частоты выборки 50 кГц:

Программа инициализации для генерации частоты 50 кГц

```
*Clkin frequency = 20 MHz, timer is running at 10 MHz.
*
LDP #0
SPLK #199,PRD     ;Загрузить период таймера 20 мкс.
OPL #8,IMR        ;Разрешить прерывание TINT в IMR.
SPLK #20h,TCR     ;Перезагрузить и перестартовать таймер.
SPLK #1000b,IFR   ;Очистеть любые ожидаемые прерывания .
CLRC INTM         ;Разрешить прерывания.
*
```

4.4.4 Программно-управляемые генераторы состояния ожидания

Программно- управляемые генераторы состояния ожидания могут увеличить цикл внешней шины до 7 машинных циклов. Это помогает взаимодействию с внешними устройствами, которые не удовлетворяют требованиям времени обращения процессора 1867ВЦ2Т (1867ВЦ2АТ). Устройства, требующие более 7 состояний ожидания, могут быть связаны, используя аппаратную линию READY.

Когда сконфигурировано нулевое состояние ожидания, останавливается внутреннее тактирование генераторов состояния ожидания, позволяя снизить энергопотребление.

Замечание: генераторы состояния ожидания влияют только на взаимодействия с внешними устройствами.

Программно-управляемые генераторы состояния ожидания управляются двумя 16-разрядными регистрами состояния ожидания (PDWSR и IOWSR) и 5-разрядным регистром управления (CWSR). Каждое из трех внешних пространств (пространства программы, данных и ввода/вывода) имеет собственное поле в программном регистре состояния ожидания (PDWSR).

4.4.4.1 Регистр состояния ожидания программ/данных

Каждое из пространств памяти программ и данных состоит из 64К адресов. Каждое 64К пространство может быть расссмотрено, как составленное из четырех блоков, содержащих по 16К слов. Каждый 16К блок в пространстве программ и данных имеет 2 отвечающих за него разряда в PDWSR, как показано в таблице 4.22. Значение 2-разрядного поля в PDWSR определяет число состояний ожидания, для каждого обращения в данном адресном диапазоне. При сбросе PDWSR устанавливается в FFFFh.

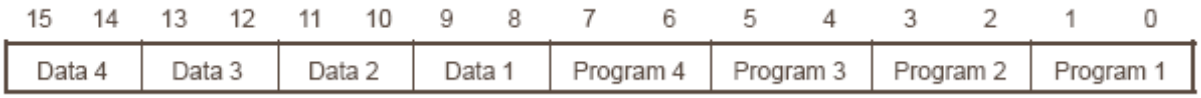


Таблица 4.22 - Границы адресов регистра состояния ожидания программ/данных

Регистр	Разряды	Пространство	Адресный диапазон
PDWSR	0 – 1	Program 1	0000h-3FFFh
	2 – 3	Program 2	4000h-7FFFh
	4 – 5	Program 3	8000h-BFFFh
	6 – 7	Program 4	C000h-FFFFh
	8 – 9	Data 1	0000h-3FFFh
	10 – 11	Data 2	4000h-7FFFh
	12 – 13	Data 3	8000h-BFFFh
	14 – 15	Data 4	C000h-FFFFh

4.4.4.2 Регистр состояния ожидания пространства ввода/вывода

Пространство ввода/вывода (I/O) состоит из 64К адресов. Регистр состояния ожидания пространства ввода/вывода (IOWSR) может задавать состояние ожидания для этого пространства двумя способами, в зависимости от значения бита BIG в контрольном регистре состояния ожидания (CWSR). Значение двухбитного поля определяет число состояний ожидания для данного порта или адресного диапазона (Таблица 4.23). При сбросе IOWSR устанавливается в FFFFh.

Если бит BIG установлен в 0, каждая из 8 пар картированных в память портов ввода/вывода связана с двухбитным полем регистра IOWSR. Значение этого двухбитного поля определяет количество циклов ожидания для каждого обращения к данному порту. Все пространство ввода/вывода сконфигурировано так, что слова объединяются попарно.
Такая конфигурация дает максимальную гибкость, когда через порты ввода/вывода обращаются к периферийным устройствам, таким как аналого/цифровые, цмфро/аналоговые устройства.

Если I/O обращается к адресуемыми устройствам (например, внешнее ОЗУ), бит BIG устанавливаться в 1. Тогда все пространство 64К делится на 8 блоков по 8К.
За каждый блок отвечает одно двухбитное поле в регистре IOWSR. Значение этого поля определяет количество циклов ожидания для обращения к конкретному блоку адресного пространства ввода/вывода.

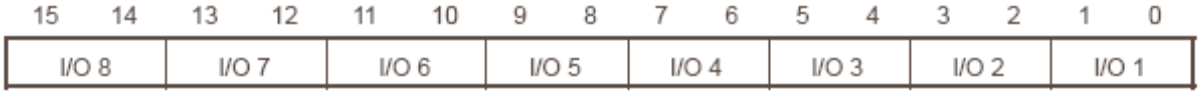


Таблица 4.23 - Границы адресов регистра состояния ожидания пространства ввода/вывода

Регистр	Разряды	Пространство ввода/вывода	Порты/16-разрядные границы адресов	
			BIG=0	BIG=1
IOWSR	0 – 1	I/O 1	Port 0/1, Port 0/1….	0000h-1FFFh
	2 – 3	I/O 2	Port 2/3, Port 12/13….	2000h-3FFFh
	4 – 5	I/O 3	Port 4/5, Port 14/15….	4000h-5FFFh
	6 – 7	I/O 4	Port 6/7, Port 16/17….	6000h-7FFFh
	8 – 9	I/O 5	Port 8/9, Port 18/19….	8000h-9FFFh
	10 – 11	I/O 6	Port 0A/0B, Port 1A/1B….	A000h-BFFFh
	12 – 13	I/O 7	Port 0C/0D, Port 1C/1D….	C000h-DFFFh
	14 – 15	I/O 8	Port 0E/0F, Port 1E/1F….	E000h-FFFFh

4.4.4.3 Контрольный регистр состояния ожидания

Контрольный регистр состояния ожидания (CWSR) позволяет выбрать одно из двух возможных применений регистра IOWSR и одно из двух возможных применений двухбитных плей регистров PDWSR и IOWSR. Поля регистра описаны в таблице 4.24.

Если соответствующий бит установлен в 0, то количество циклов ожиданий для внешних обращений в данном пространстве равно значению двухбитного поля состояния ожидания. Если соответствующий бит установлен в 1, то состояние ожидания определяется значениями двухбитных полей по таблице 4.25 Для исключения конфигурируемости памяти со слишком малыми состояниями ожидания в процессе установки регистров состояния ожидания, сначала устанавливается CWSR, а затем конфигурируются PDWSR и IOWSR.

Таблица 4.24 - Описание полей контрольного регистра состояния ожидания (CWSR)

Бит	Название	Значение при сбросе	Описание функционирования
1	2	3	4
15-5	Зарезервировано	0	Зарезервированы.
4	BIG	0	Определяет работу IOWSR: BIG=0 Все пространство I/O портов делится на 8 групп, каждой из которых соответствует двухбитное поле регистра IOWSR. BIG=1 Все адресное пространство I/O портов делится на 8 блоков по 8К, каждому из которых соответствует двухбитное поле регистра IOWSR.
3	I/O High	1	Определяет, как работает двухбитное поле регистра IOWSR для старшей части пространства ввода/вывода (I/O5 - I/O8) I/O High=0 Количество состояний ожидания для старшей половины пространства ввода/вывода будут 0,1,2 или 3. I/O High=1 Количество состояний ожидания для старшей половины пространства ввода/вывода будут 0,1,3 или 7.
2	I/O Low	1	Определяет, как работает двухбитное поле регистра IOWSR для младшей части пространства ввода/вывода (I/O1 - I/O4) I/O Low=0 Количество состояний ожидания для младшей половины пространства ввода/вывода будут 0,1,2 или 3. I/O Low=1 Количество состояний ожидания для младшей половины пространства ввода/вывода будут 0,1,3 или 7.

Продолжение таблицы 4.24

1	2	3	4
15-5	Зарезервировано	0	Зарезервированы.
4	BIG	0	Определяет работу IOWSR: BIG=0 Все пространство I/O портов делится на 8 групп, каждой из которых соответствует двухбитное поле регистра IOWSR. BIG=1 Все адресное пространство I/O портов делится на 8 блоков по 8К, каждому из которых соответствует двухбитное поле регистра IOWSR.
3	I/O High	1	Определяет, как работает двухбитное поле регистра IOWSR для старшей части пространства ввода/вывода (I/O5 - I/O8) I/O High=0 Количество состояний ожидания для старшей половины пространства ввода/вывода будут 0,1,2 или 3. I/O High=1 Количество состояний ожидания для старшей половины пространства ввода/вывода будут 0,1,3 или 7.
2	I/O Low	1	Определяет, как работает двухбитное поле регистра IOWSR для младшей части пространства ввода/вывода (I/O1 - I/O4) I/O Low=0 Количество состояний ожидания для младшей половины пространства ввода/вывода будут 0,1,2 или 3. I/O Low=1 Количество состояний ожидания для младшей половины пространства ввода/вывода будут 0,1,3 или 7.
1	D	1	Бит пространства памяти данных. Определяет, как работают двухбитные поля регтстра PDWSR для определения состояний ожиданий для пространства памяти данных. D=0 Количество состояний ожидания для пространства памяти данных будут 0,1,2,3 D=1 Количество состояний ожидания для пространства памяти данных будут 0,1,3,7
0	P	1	Бит пространства памяти программ. Определяет, как работают двухбитные поля регистра PDWSR для определения количества состояний ожидания при обращении к внешней памяти программ. P=0 Количество состояний ожидания для пространства памяти программ будут 0,1,2,3 P=1 Количество состояний ожидания для пространства памяти программ будут 0,1,3,7

Таблица 4.25 - Значения двухбитных полей и количество циклов ожидания как функция битаов (0-3) регистра CWSR

Двухбитное поле PDWSR или IOWSR (двоичное значение)	Количество циклов ожидания (биты 0-3, CWSR = 0)	Количество циклов ожидания (биты 0-3, CWSR = 1)
00	0	0
01	1	1
10	2	3
11	3	7

На рисунке 4.36 показана блок-диаграмма генератора состояния ожидания для пространства внешней памяти программ. Когда декодируется доступ к внешней памяти программ, соответствующее поле регистра состояния ожидания PDWSR загружается в счетчик. Если поле не 000_2, сигнал «не готов» посылается в ЦПУ. Состояние «не готов» поддерживается до тех пор, пока счетчик не уменьшится до нуля и внешняя линия READY не станет высокой. Внешний сигнал READY и READY от состояния ожидания складываются по OR и формируют сигнал $\overline{WAIT}$. Внешний READY анализируется по переднему фронту CLKOUT1 .Если используется генератор состояний ожидания, внешний READY анализируется на последнем цикле ожидания.

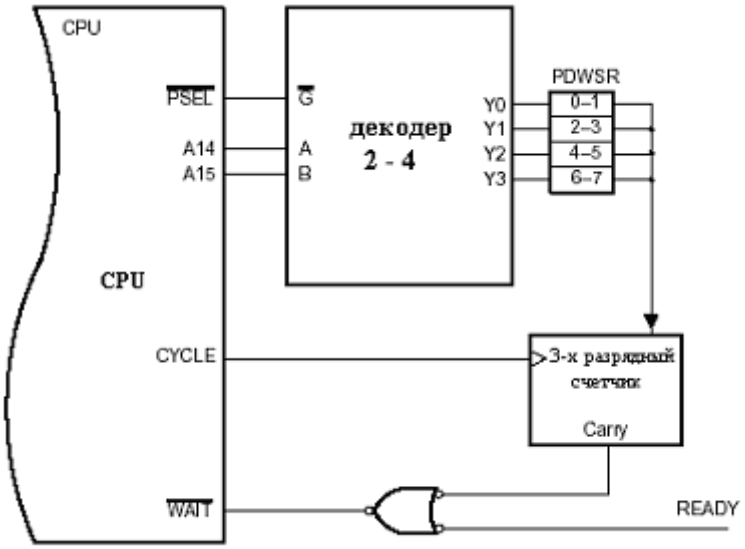


Рисунок 4.36 - Блок-диаграмма программноуправляемого генератора состояния ожидания

4.4.5 Контакты ввода/вывода общего назначения

Процессор 1867ВЦ2Т (1867ВЦ2АТ) имеет два вывода общего назначения, которые являются программно-управляемыми. Вход $\overline{BIO}$ - вход управления переходами, а XF - выход внешнего флага.

Вход управления переходами ($\overline{BIO}$) пригоден для наблюдения состояния периферийных устройств, например как альтернатива для использования прерывания. Переход может выполняться условно, в зависимости от состояния $\overline{BIO}$. Временная диаграмма, показанная на рисунке 4.37, показывает как работает $\overline{BIO}$. Эта временная диаграмма для последовательности одноцикловых, однословных инструкций, размещенных во внешней памяти.

Если используется инструкция XC, то условие $\overline{BIO}$ проверяется в фазе декодирования (2-фаза) конвейера. Все остальные инструкции (BCND, BCNDD, CC, CCD, RETC, RETCD) проверяют $\overline{BIO}$ на фазе выполнения (4-фаза) конвейера.

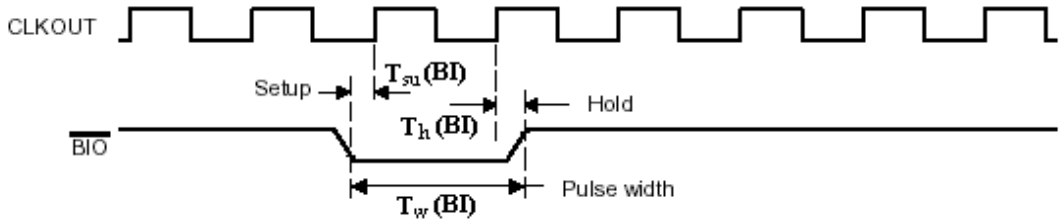


Рисунок 4.37 - Диаграмма синхронизации $\overline{BIO}$

Выход XF (внешний флаг) к внешним устройствам програмно управляем. XF устанавливается в высокий уровень инструкцией SETC XF (установить внешний флаг) и сбрасывается в нижний уровень инструкцией CLRC XF (сброс внешнего флага). XF устанавливается высоким после сброса устройства. На рисунке 4.38 показано временное соотношение между временем заквата инструкций SETC/CLRC и временем установки или сброса XF. Это временная диаграмма последовательности одноцикловых и однословных инструкций, размещенных во внешней памяти. Фактическая временная диаграмма может отлмчаться от приведенной при последовательностях других инструкций.

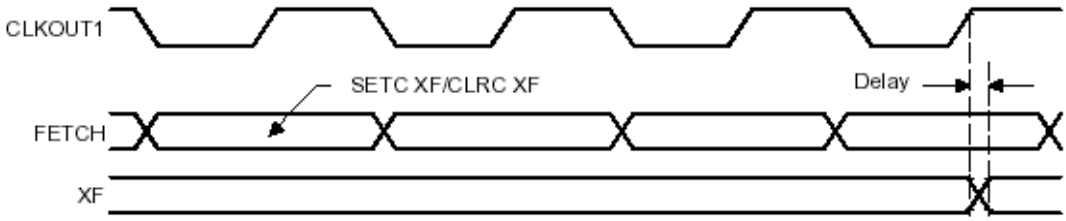


Рисунок 4.38 - Диаграмма синхронизации внешнего флага

4.4.6 Порты ввода/вывода

Устройство 1867ВЦ2Т (1867ВЦ2АТ) имеет 64К параллельных портов ввода/вывода (I/O). 16 из них картированы памяти на странице 0, как показано в таблице 4.2. Можно достичь 64К портов ввода/вывода, используя инструкции IN и OUT или любую инструкцию, которая пишет или читает в (из) пространства памяти данных. Доступ в пространство ввода/вывода отличается от доступа в память программ и данных тем, что сигнал $\overline{IS}$ падает в низкий уровень; сигнал $\overline{DS}$ неактивен, даже если через порты ввода/вывода идет обращение к памяти данных.

Сигнал $\overline{RD}$ может быть использован совместно с логикой выбора кристалла для выработки сигнала «готов выводить» для внешних периферийных устройств. Сигнал $\overline{WE}$ может быть использован совместно с логикой выбора кристалла для выработки сигнала «готов записывать» для внешних периферийных устройств. Рисунок 4.39 демонстрирует типичную схему интерфейса портов ввода/вывода.

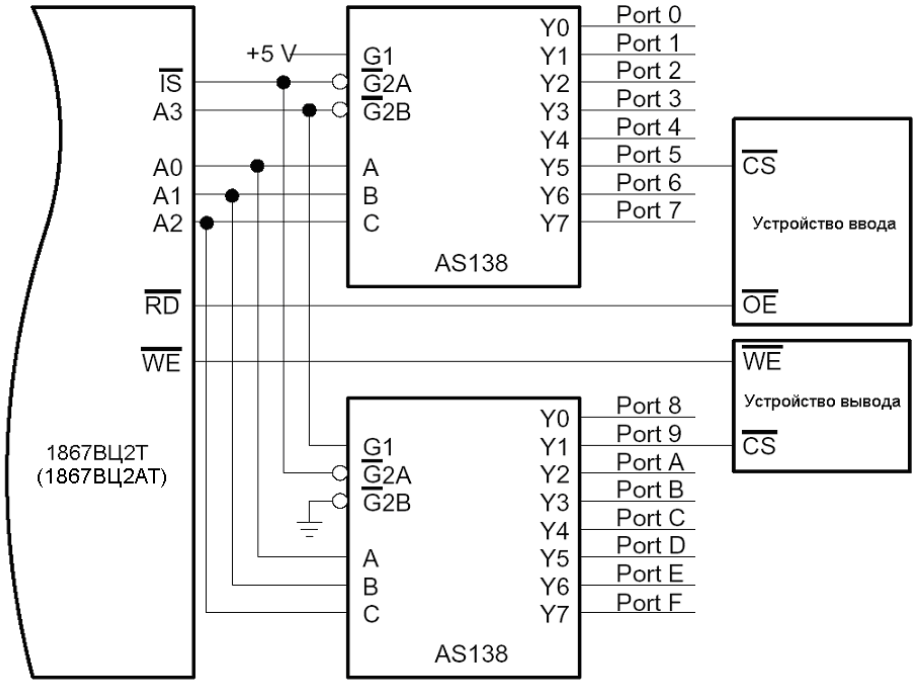


Рисунок 4.39 - Схема интерфейса порта I/O

4.4.7 Последовательный порт

Последовательный порт обеспечивает прямую двунаправленную связь с устройствами, такими как кодеки, последовательные A/D конвертеры и другими последовательными системами. Сигналы интерфейса последовательного порта совместимы со многими стандартными кодеками и другими последовательными устройствами. Последовательный порт может также использоваться для микропроцессорных соединений между процессорами в мультипроцессорных системах.

В процессоре 1867ВЦ2Т (1867ВЦ2АТ) предусмотрено 2 интерфейса последовательного порта. Стандартный интерфейс последовательного порта (SP) и интерфейс последовательного порта с разделением по времени (TDM). TDM порт может работать в TDM - режиме или в не TDM - режиме. Когда TDM порт работает не в TDM режиме (или стандатном), то он работает как обычный последовательный порт (SP) как описано в этом разделе.

В последовательном порты операции приема и передачи являются дважды буферизованными, что позволяет осуществлять так называемый режим продолжительной передачи или приема по 8 или 16 бит. Этот режим позволяет передавать пакет информации, где требуется только один (FSR или FSX) импульс в начале пакета и не требуется их передавать перед каждым словом пакета.

Последовательный порт - статическое устройство и работает на более низкой частоте, чем процессор. Максимальная частота рыботы последовательного порта составляет ¼ от частоты CLKOUT1 и может быть при использовании режима внутреннено тактирования последовательного порта. При сбросе последовательного порта, можно для уменьшения энергопотребления отключить внутреннее тактирование портов.

4.4.7.1 Регистры интерфейса последовательного порта

Последовательный порт использует в работе три картированных в памяти регистра (SPC, DXR, DRR) и два других программно-недоступных, но участвкющих в двойной буфферизации регистра (RSR, XSR). Регистры последовательного порта представлены в таблице 4.26.

Биты, контакты и регистры, которые управляют работой последовательного порта, перечисляются в таблице 4.23.

Таблица 4.26 - Регистры последовательного порта

	Регистры	Описание
0020h	DRR	Регистр приема данных
0021h	DXR	Регистр передачи данных
0022h	SPC	Регистр управления последовательным портом
-	RSR	Приемный сдвиговый регистр
-	XSR	Передающий сдвиговый регистр

Регистр приема данных (DRR) - 16-разрядный картированный в памяти регистр приема данных захватывает приходящие последоательно данные из сдвигового регистра RSR для последующей записи на внутреннюю шину данных. При сбросе DRR обнуляется.

Регистр передачи данных (DXR) - 16-разрядный картированный в памяти регистр передачи данных захватывает данные с внутренней шины данных для загрузки их в XSR регистр и последующей передачи. При сбросе DXR обнуляется.

Регистр управления последовательным портом (SPC) - 16-разрядный картированный в памяти регистр управления последовательным портом содержит биты управления режимом и биты состояния последовательного порта.

Регистр сдвига принимаемых данных (RSR) - 16-разрядный сдвиговый регистр приема данных захватывает данные, приходящие со входа приема данных (DR) и управляет передачей этих данных в DRR регистр.

Регистр сдвига передаваемых данных (XSR) - 16-разрядный сдвиговый регистр передаваемых данных управляет передачей этих данных из DXR в XSR регистр и передает их на выход DX.

В обычном режиме работы последовательного порта DXR программно загружается данными, и его содержимое периодически считывается в сдвиовый регистр и передается наружу, когда дана команда на передачу. DRR автоматически загружается данными, принятыми последовательным портом и программно считывается из DRR.

Если регистры DXR и DRR в данный момент не используются для работы последовательного порта, то они могут использоваться как картированные в памяти регистры общего назначения для хранения информации. Если эти регистры будут использоваться как обычные регистры, следует отключить передающие/принимающие регистры порта путем подачи на внешние выводы порта неактивных уровней или при помощи сброса порта.

4.4.7.2 Работа интерфейса последовательного порта

Этот раздел описывает стандартный режим работы последовательного порта, включая TDM порт в режиме последовательного порта. В таблице 4.27 приведены внешние выводы, используемые для работы последовательного порта. На рисунке 4.40 покозано соединение двух процессоров для однонаправленной передачи от 0-го к 1-му.

Передающий и принимающий процессор связаны тремя сигналами. Передаваемые данные подаются на выход DX. Синхронизирующий импульс фрейма передачи (FSX) показывает начало передачи (в начале пакета данных). Сигнал тактирования передачи (CLKX) тактирует побитную передачу. Соответствующие выводы приемного процессора - DR, FSR, CLR соответственно.

Таблица 4.27 - Выводы последовательного порта

Имя	Описание
IN0	Текущий уровень на CLKR
IN1	Текущий уровень на CLKX
CLKX	Тактирующий сигнал передачи
CLKR	Тактирующий сигнал приема
DX	Сигнал последовательной передачи данных
DR	Сигнал последовательного приема данных
FSX	Сигнал синхронизации фрейма передачи данных
FSR	Сигнал синхронизации фрейма приема данных

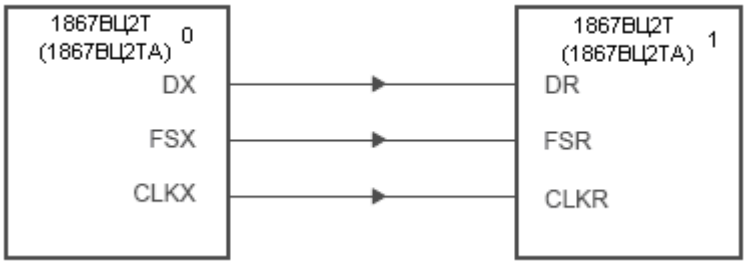


Рисунок 4.40 - Однонаправленная передача по последовательному порту.

Данные для передачи записываются в DXR, а принятые данные читаются из DRR. Начало передачи обозначается записью данных в DXR, данные копируются в XSR когда он пуст (когда последнее слово последовательно передалось через вывод DX). XSR управляет сдвигом данных на DX, таким образом разрешая следующую запись в DXR как только завершится копирование из DXR в XSR.

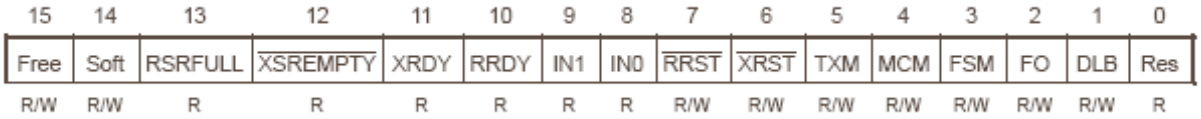
В процессе передачи, по завершению копирования из DXR в XSR, значение бита готовности передачи (XRDY) в SPC меняется из 0 в 1. Как только происходит это изменение, вырабатывается прерывание передачи последовательного порта (XINT), которое сигнализирует процессору о том, что DXR готов снова загрузиться.

Также происходит и с принимающим процессором. Данные с вывода DR сдвигаются в RSR, затем копируются в DRR, из которого могут быть считаны. По завершении копирования RSR в DRR значение бита готовности приема (RRDY в SPC) меняется из 0 в 1. Как только происходит это изменение, вырабатывается прерывание по приему последовательного порта (PINT). Порт обладает двойной буферизацией, так как он

может передавать или принимать из или в DXR или DRR в то время, как уже происходит передача или прием. Передача синхронизируется при помощи FSX в прерывном режиме.

4.4.7.3 Установка конфигурации последовательного порта

Регистр SPC содержит биты управления последовательным портом. На рисунке 4.41 показаны поля битов регистра SPC. Эти биты могут быть установлены, очищены, переключены или загружены, используя инструкции PLU.
Таблица 4.28 описывает бты регистра управления последовательным портом.



R - доступен для чтения;
R/W - доступен для чтения / записи;

Рисунок 4.41 - Регистр управления последовательным портом (SPC)

Таблица 4.28 - Описание битов регистра управления последовательным портом

Бит	Имя	Значение при сбросе	Описание функционирования
1	2	3	4
15	FREE	0	Используется совместно с битом SOFT для определения состояния тактирования последовательного порта, когда фиксируется остановка. FREE=0 - Бит SOFT выбирает режим эмуляции FREE=1 - Тактирование продолжает работать независимо от бита SOFT
14	SOFT	0	Используется совместно с битом FREE для определения состояния тактирования последовательного порта, когда фиксируется остановка. Когда бит FREE очищен, то SOFT выбирает режим эмуляции SOFT=0 - Тактирование останавливается, прерывая всякую передачу SOFT=1 - Тактирование прекращается по завершению текущей передачи.

Продолжение таблицы 4.28

1	2	3	4
13	RSRFULL	0	Бит заполнения приемного сдвигового регистра. Используется, когда происходит переполнение приемного сдвигового регистра RSR. Переполнение происходит, когда RSR полный, а DRR не был считан после последней загрузки из RSR. В последовательном порту при FSM=1, RSRFULL может стать 1 только если сформировался импульс FSR. Если FSM=0, RSRFULL не ждет импульса FSR, а становится 1, когда происходит переполнение. RSRFULL=0 Любое из следующих трез событий очищает RSRFULL в 0: - чтение DRR; - сброс приемника (= 0); - сброс всего устройства.
12	$\overline{XRESEMPTY}$	0	Бит, показывающий, что передающий сдвиговый регистр пуст. Показывет, что не произошла своевременная загрузка XSR. Это происходит, когда XSR пуст и DXR не был загружен после последнего чтения из DXR в XSR. $\overline{XRESEMPTY}$ =0 Любое из следующих трех событий очищает $\overline{XRESEMPTY}$ в 0: - не произошла своевременная загрузка; - сброс передатчика ($\overline{XRST}$ = 0); - сброс всего устройства. $\overline{XRESEMPTY}$ =1 В последовательном порту $\overline{XRESEMPTY}$ деактивируется ($\overline{XRESEMPTY}$ =1) сразу после записи в DXR.
11	XRDY	1	Бит готовности передачи. Переход из 0 в 1 бита XRDY показывает, что содержимое DXR было скопировано в XSR и DXR готов к новой загрузке. Прерывание по передаче вырабатывается после этого перехода. При сбросе процессора или при сбросе передатчика последовательного порта ($\overline{XRST}$ =0), XRDY становится в 1.
10	RRDY	0	Бит готовности приема данных. Переход из 0 в 1 бита RRDY показывает, что содержимое RSR было скопировано в DRR и данные могут считываться из него. Прерывание по приему вырабатывается после этого перехода. Этот бит может быть опрошен прграммой не зависимо от использования прерываний последовательного порта. При сбросе процессора или при сбросе приемника последовательного порта ($\overline{RRST}$=0), XRDY становится в 0

Продолжение таблицы 4.28

1	2	3	4
9	IN1	X	Вход1. Позволяет использовать внешний вывод CLKX как битовый вход. IN1 отражает текущий уровень вывода CLKX. При переключении уровней CLKX существует задержка 0,5-1,5 цикла CLKOUT1, пока новое значение CLKX будет представлено в SPC.
8	IN0	X	Вход0. Позволяет использовать внешний вывод CLKR как битовый вход. . IN0 отражает текущий уровень вывода CLKR. При переключении уровней CLKX существует задержка 0,5-1,5 цикла CLKOUT1, пока новое значение CLKR будет представлено в SPC.
7	$\overline{RRST}$	0	Бит сброса приемника. Разрешает и прекращает работу приемника последовательного порта. $\overline{RRST}$=0 - Сброс приемного последовательного порта. Запись 0 в $\overline{RRST}$ очищает биты RSRFULL и RRDY в 0. $\overline{RRST}$=1 - Приемный последовательный порт задействован.
6	$\overline{XRST}$	0	Бит сброса передатчика. Разрешает и прекращает работу передатчика последовательного порта. $\overline{XRST}$=0 - Сброс передающего последовательного порта. Если бит XRDY был 0, запись 0 в $\overline{XRST}$ вырабатывает прерывание по передаче. Запись 0 в $\overline{XRST}$ очищает бит $\overline{XRESEMPTY}$ в 0 и устанавливает бит XRDY в 1. $\overline{XRST}$=0 - сброс передающего последовательного порта; $\overline{XRST}$=1 - Передающей последовательный порт задействован.
5	TXM	0	Бит режима передачи. Конфигурирует вывод FSX как вход (TXM=0) или как выход (TXM=1). TXM=0 - Внешняя синхронизация фрейма. Передающий порт ждет внешнего импульса FSX TXM=1 - Внутренняя синхронизация фрейма. FSX вырабатывается внутренне, когда данные передаются из DRX в XSR для обозначения передачи данных. Импульсы FSX синхронизированы с CLKX.

Продолжение таблицы 4.28

1	2	3	4
4	MCM	0	Бит режима тактирования. Определяет источник тактов для CLKX. MCM=0 - CLKX берется с внешнего вывода CLKX; MCM=1 - CLKX вырабатывается внутри процессора с частотой, равной ¼ частоты CLKOUT1. Если MCM=1 и DLB=1 CLKR тоже вырабатывается внутри процессора.
3	FSM	0	Бит режима синхронизации фреймов. Определяет, нужны ли импульсы FSX, FSR после первого импульса инициализации. FSM=0 - Непрерывный режим. Синхроимпульсы не нужны за исключением первого, но они не игнорируются, поэтому невовремя поданные импульсы могут стать причиной ошибок в последовательной передаче/приеме данных; FSM=1 - Прерывный режим. Для передачи/приема Каждого слова требуется импульс FSX/FSR.
2	FO	0	Бит фармата данных. Определяет длину передаваемых/принимаемых слов. FO=0 - данные передаются/принимаются как 16-разрядные слова; FO=1 - Данные передаются по 8-разрядов. Первыми передаются старшие разряды (MSB).
1	DLB	0	Бит режима внутренней петли. Используется для установки работы порта в режиме внутренней петли. DLB=0 - Режим внутренней петли отключен. DR, FSR, CLKR берутся с соответствующих внешних выводов; DLB=1 - Задействован режим внутренней петли. DR и FSR подаются на DX и FSX соответственно. CLKR управляется CLKX если MCM=1. Если DLB=1 и MCM=0 сигнал CLKR берется с внешнего вывода CLKR. Эта конфигурация позволяет CLKR и CLKX тактироваться внешним источником. В режиме DLB сигналы FSX и DX появляются на внешних выводах, а FSR и DR нет. В режиме DLB могут использоваться как внутренние, так и внешние сигналы FSX, как определено битом TXM.
0	RES	0	Бит зарезервирован. Для последовательного порта всегда читается как 0. Этот бит имеет свою функцию в TDM порту.

Таблица 4.29 - Конфигурация тактирования последовательного порта

FREE	SOFT	Конфигурация тактирования.
0	0	Немедленная остановка тактирования.
0	1	Передающий порт останавливается по завершении
1	X	слова. На приемный порт не влияет. Последовательный порт продолжает работать.

4.4.7.4 Передача и прием данных в прерывном режиме

В прерывном режиме есть периоды, когда порт неактивен между передачами слов. Передача каждого слова обозначается синхроимпульсом FSX. На передающем устройстве командой передавать служит команда записи в DXR. Затем данные, записанные в DXR передаются в XSR и, после синхроимпульса FSX (выработанного внутренне или внешне в зависимости от TXM) данне из XSR на вывод DX. Заметим, что в последовательном порту переход данных из DXR в XSR происходит на втором такте CLKX после того, как DXR был загружен по переднему фронту. Как только загружается XSR значением из DXR, XRDY переходит в 1, вырабатывая прерывание по передаче (XINT) и устанавливая $\overline{XRESEMPTY}$ в 1.

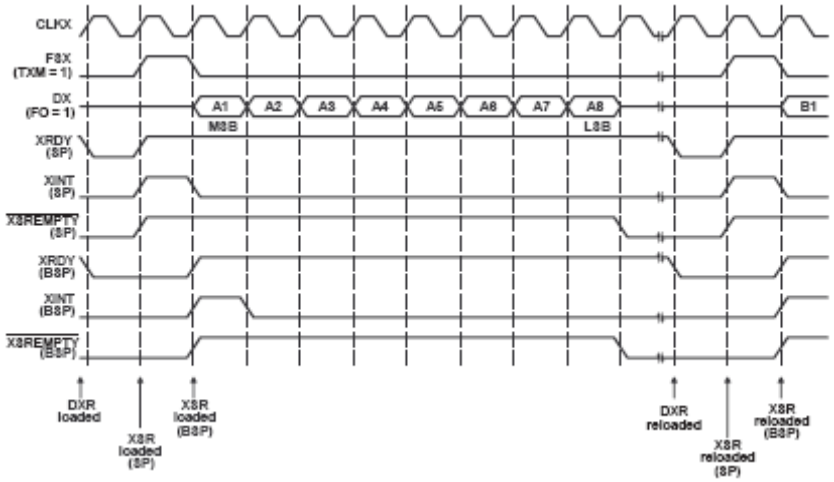


Рисунок 4.42 - Режим передачи последовательного порта в прерывном режиме

Заметим, что запись из DXR в XSR происходит только, если XSR пуст и DXR загрузился после предыдущего считывания из DXR в XSR. Если DXR перезагружается до того, как его старое содержимое было переданов XSR, то его предыдущее значение переписывается. Поэтому эсли мы хотим перезагрузить DXR, то делать это нужно, когда XRDY=1, то есть или после прерывания по передаче или опросив бит XRDY.

Временные диаграммы для внутренней (TXM=1, FSX - выход) и внешней (TXM=0, FSX - вход) генерации синхроимпульсов FSX отличаются. Это различие происходит

потому, что в первом случае синхроимпуль FSX вырабатывается как результат записи в DXR, а во втором случае устройство должно записать в DXR и ждать внешнего синхроимпульса.

Если выбран режим внутренней синхронизации фреймов (TXM=1), то импульс FSX вырабатывается по переднему фронту 2-го такта CLKX после записи в DXR. Если выбран режим внутренней синхронизации фреймов, то события, описанные выше произойдут как только придет внешний импульс FSX.

По следующему переднему фронту CLKX после того, как FSX стал высоким, первый бит данных (MSB) выходит на вывод DX. Так, если FSX вырабатывется внутренне, (TXM=1) необходимо 2 такта CLKX от момента загрузки в DXR до выхода на линию DX. Если FSX вырабатывается внешне, то после загрузки DXR порт ждет этого импульса. После того, как FSX падает в 0, сдвигаются остальные биты данных. Когда переданы все биты, DX становится в 3-е состояние. По окончании каждой передачи, если DXR не был перезагружен когда выработалось прерывание XINT, $\overline{XRESEMPTY}$ становится активным (низким), показывая, что DXR своевременно не загружен. Если синхронизация фреймов внешняя, $\overline{XRESEMPTY}$ активный, пришел внешний импульс FSX, то будут переданы старые данные, находящиеся в DXR.

Заметим, что длина первого бита передаваемых данных может быть различной при внешнем FSX (может быть длиннее одног такта CLKX). Если FSX вырабатывется внутренне, то длина всех битов будет рана длине такта CLKX.

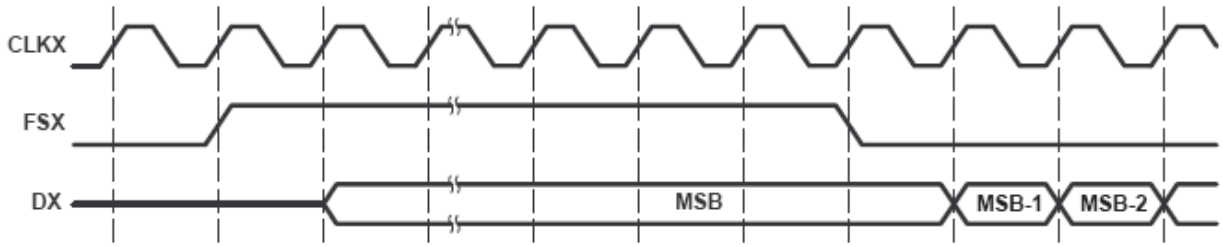


Рисунок 4.43 - Передача по последовательному порту с длинным импульсом FSX.

Передающий последовательный порт с внешними синхроимпульсами FSX работает также, как и с внутренними за исключением того, что передача не начнется пока не пришел внешний импульс FSX. Если он придет спустя много тактов CLKX после того как DXR был загружен, двойной буфер будет занят и останется в таком состоянии, пока не придет импульс FSX.

В процессе операции приема, сдвиг в RSR начинается по заднему фронту CLKR, после того как FSR стал низким. Затем, после того,как последние данные приняты, содержимое RSR передается в DRR по заднему фронту CLKR, и RRDY переходит в высокое состояние, вырабатывя прерывание по приему (RINT).

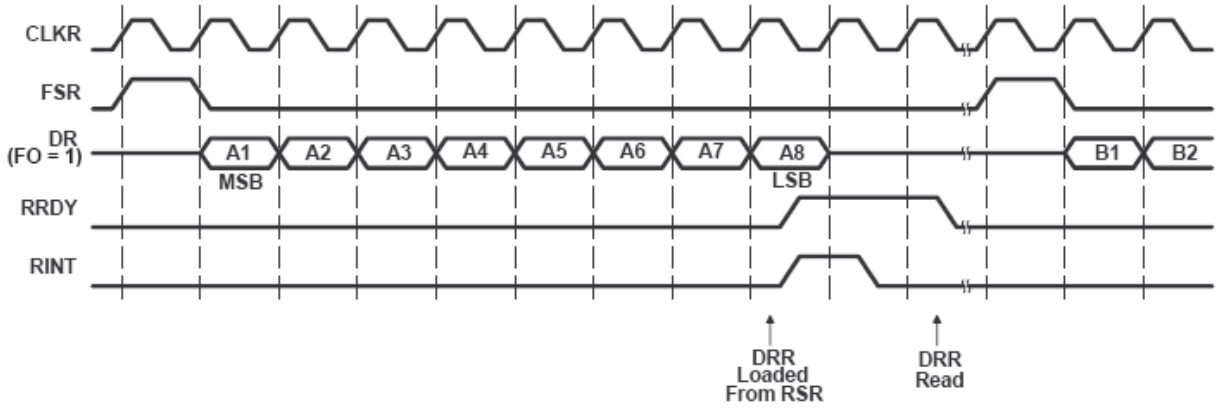


Рисунок 4.44 - Прием по последовательному порту в прерывном режиме.

Если DRR не был считан после предыдущего приема, и принято следующее слово, то, пока DRR и RSR оба полные не будет принято ни одного бита данных. В этом случае бит RSRFULL выставляется в 1.

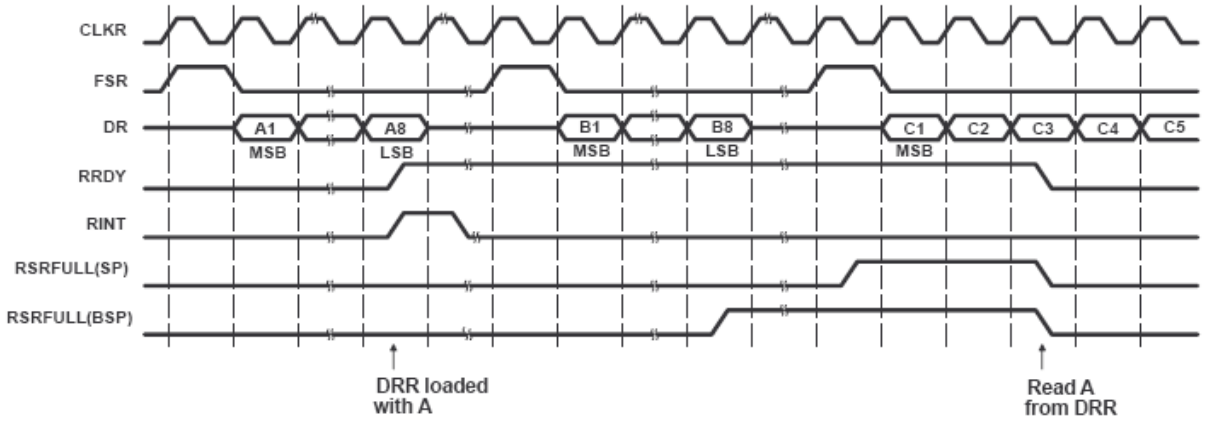


Рисунок 4.45 - Переполнение при приеме по последовательному порту в прерывном режиме.

Переполнение DRR может привести к ошибке. Следующая информация, которая должна быть принята, будет утеряна.

В последовательном порту содержимое DSR защищено от переполнения. Данные начинают теряться, когда DRR и FSR заполнены и приходит следующий импульс FSR и RSRFULL становится в 1. Потерю данных можно предотвратить, если программно опрашивать бит RSRFULL и считывать DRR сразу после того, как RSRFULL становится

в 1. Это возможно только тогда, когда частота CLKR ниже, чем CLKOUT1. RSRFULL устанавливается в 1 по заднему фронту CLKR при высоком FSR и данные начинают приниматься по следующему переднему фронту CLKR т. е. времени на опрос бита RSRFULL и считывания DRR остается только половина такта CLKR. Если импульсы FSX для приемного порта значительно длиннее, чем 1 такт CLKR, то временная диаграмма приема похожа на временную диаграмму передачи с длинными импульсами FSX. Однако прием всех битов, включая первый, просто задерживается на время, пока FRS не станет низким (рисунок 4.46).

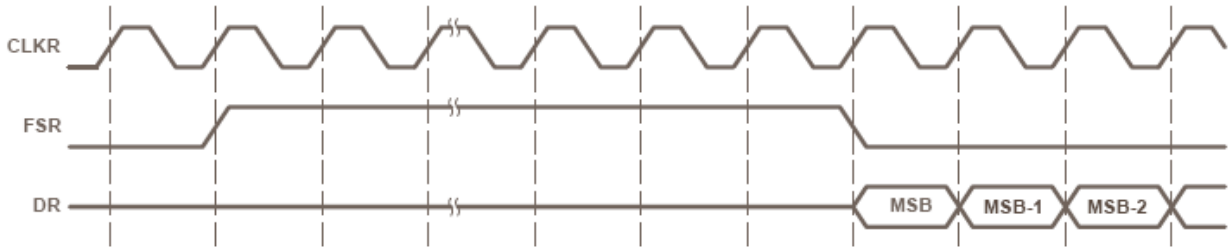


Рисунок 4.46 - Прием по последовательному порту с длинным импульсом FSR.

Заметим, что если частота передачи слов возрастает, период неактивности между смежными передачами слов уменьшается до нуля. Это относится к минимальному периоду между импульсами FSX (FSR), что соответствует 8- или 16- тактам CLKX/CLKR (в зависимости от F0), иэто будет максимальной частотой, с которой последовательный порт может работать. Временная диаграмма передачи максимальной частотой есть сжатая версия диаграммы на рисунке 4.42 , как показано на рисунке 4.47.

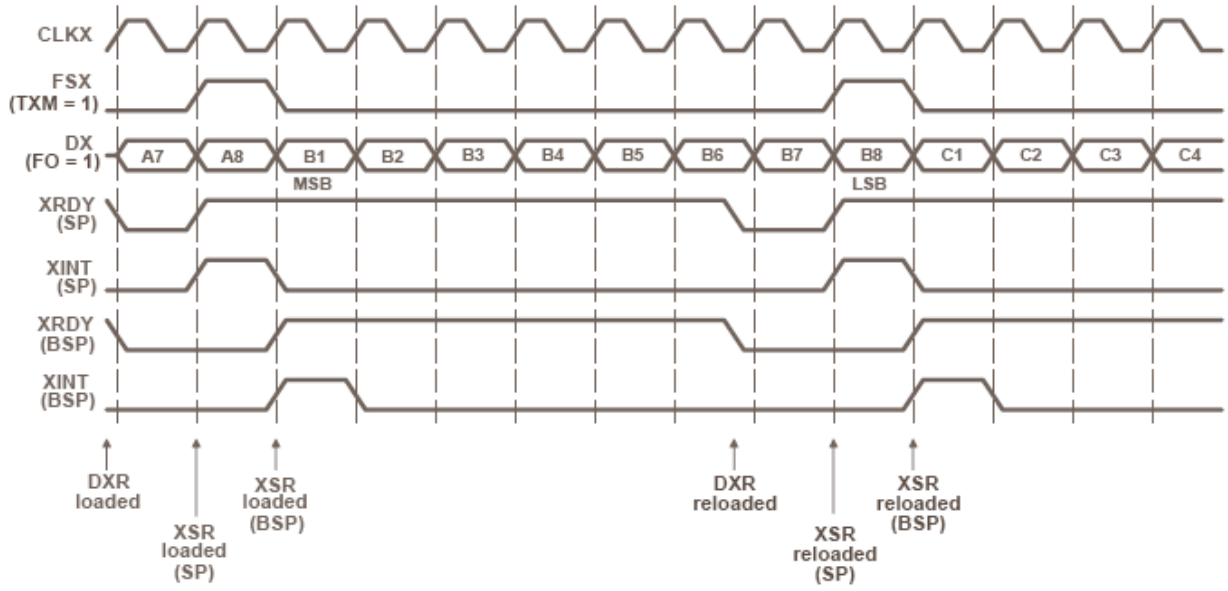


Рисунок 4.47 - Передача по последовательному порту в прерывном режиме с максимальной частотой слов (пакетов данных).

На максимальной частоте передачи слов, биты в соседних словах передаются вплотную друг к другу без промежутков между ними. Импульс FSX(FSR) перекрываер собой последний бит переданного предыдущего слова. Временная диаграмма приема на максимальной частоте слов показана на рисунке 4.48

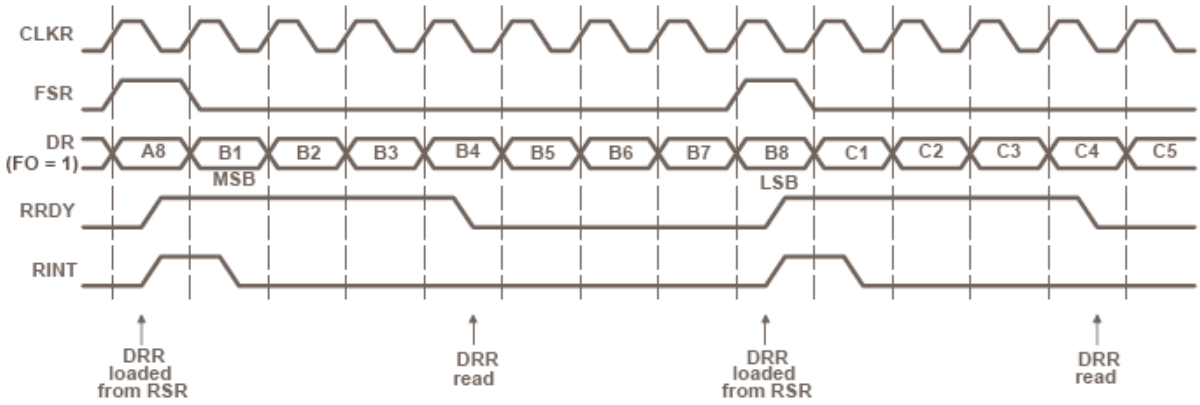


Рисунок 4.48 - Прием по последовательному порту в прерывном режиме с максимальной частотой слов (пакетов данных).

Как показано на рисунках 4.47, 4.48 для передачи данных на максимальной частоте слов в прерывном режиме, слова передаются с постоянной скоростью и сигналы CLKX, CLKR последовательного порта дают достаточно временной информации для обеспечивания последовательного потока данных. Получается, что импульсы синхронизации фреймов являются излишними. Следовательно теоретически для инициализации передачи пакета слов требуется только один начальный импульс FSX(FSR). В процессоре 1867ВЦ2Т (1867ВЦ2АТ) есть такой режим, называемый непрерывным.

4.4.7.5 Передача и прием данных в непрерывном режиме

В непрерывном режиме нет необходимости в импульсах FSX(FSR) для передачи (приема) каждого слова на максимальной частоте пакетов данных кроме начального импульса инициализации передачи. Непрерывный режим выбирается путем установки бита FSM в 0. Заметим, что если FSM=0, импульсы синхронизации фреймов не требуются, но не игнарируются, поэтому если они будут поданы неправельно, то возможны ошибки в последовательном приеме/передаче.

В непрерывном режиме обрабатывается только 1 импульс FSX следом за первой загрузкой DXR, и больше не вырабатывается ни одного импульса. И до тех пор, пока в DXR, будут загружаться новые слова, непрерывная передача будет продолжаться. Если не происходит загрузки DXR, то передача останавливается, что в случае непрерывного режима соответствует $\overline{XRESEMPTY}$ =0. Если DXR снова загружается после остановки

передачи, порт снова начинает передавать в непрерывном режиме и вырабатывается один импульс FSX при условии, что выбран режим внутренней синхронизации фреймов.

Разница междурежимами внутренней и внешней синхронизации фреймов такая же, как и для прерывного режима. Если выбран режим внешней синхронизации, то, когда DXR загружен, появление внешнего импульса FSX обозначает начало непрерывного режима передачи. Непрерывный режим передачи может быть отменен (и включен прерывный) только путем реконфигурации и сброса последовательного порта. Простое изменение бита FSM в процессе передачи или остановка не переключают порт в непрерывный режим.

Временная диаграмма непрерывного режима передачи, показанная на рисунке 4.49, похожа на диаграмму передачи с максимальной частотой пакетов данных в прерывном режиме, как показано на рисунке 4.47. Основное отличие заключаетя в отсутствии импульсов синхронизации фреймов после первого. До тех пор, пока в DXR будут загружаться новые слова, этот режим будет продолжать работать. Перезагрузка DXR происходит также, как и в прерывном режиме. Новый внешний импульс FSX прервет текущую передачу, что вызовет потерю одного пакета данных и проинициализирует только начало нового непрерывного режима передачи.

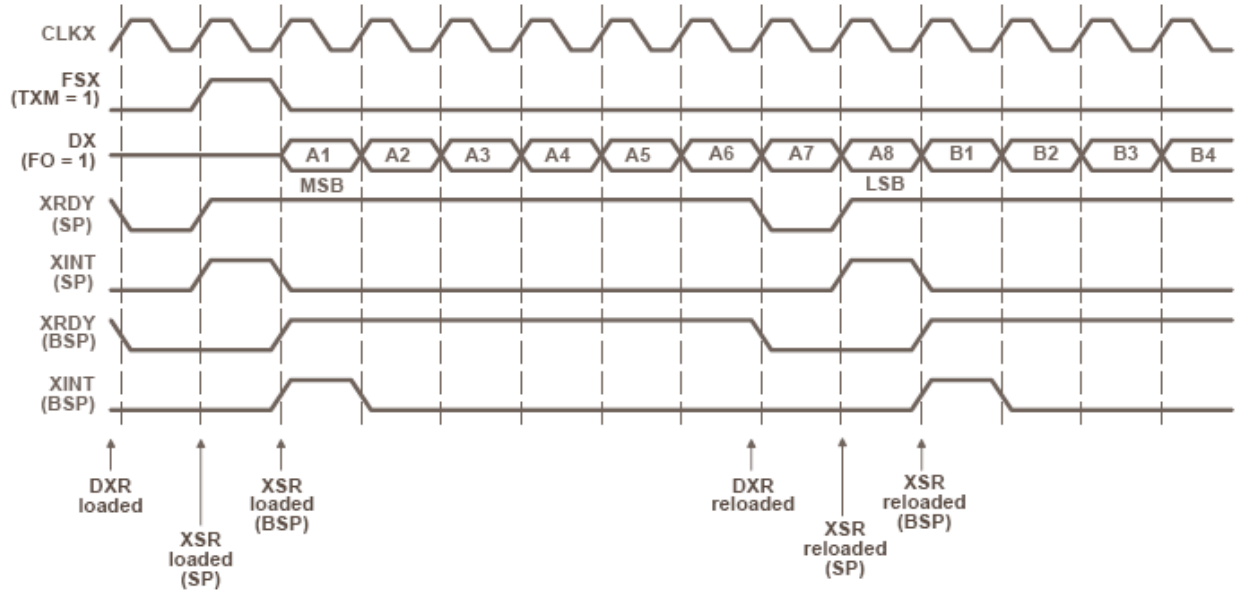


Рисунок 4.49 - Передача по последовательному порту в непрерывном режиме.

Прием данных в непрерывном режиме такой же, как и передача. После инициализирующего импульса на FSR, больше импульсов не требуется. Этот рнжим будет продолжаться до тех пор, пока DRR считывается после каждой передачи. Если DRR не считывается по окончанию следующей передачи, то процесс приема остановится и выставится бит RSRFULL, показывая переполнение.

Если произошло переполнение, то считывание из DRR снова запустит непрерывный режим, начиная со следующего слова/байта после того, как DRR был прочитан; нового импульса FSR не требуется.

Непрерывный режим приема может быть отключен только путем переконфигурации и сбросом последовательного порта. Простым изменением бита FSM или остановкой нельзя переключить порт в прерывный режим приема. Временная диаграмма непрерывного режима приема показана на рисунке 4.50.

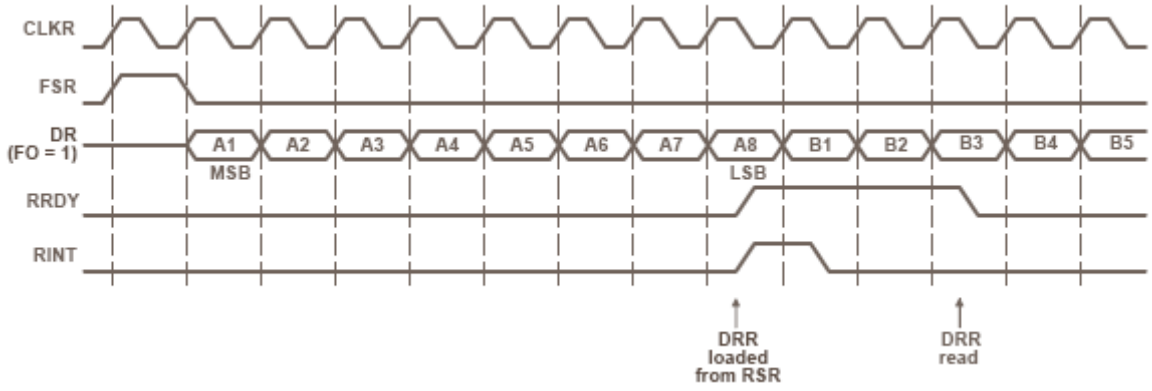


Рисунок 4.50 - Прием по последовательному порту в непрерывном режиме.

На рисунке 4.50 показан только 1 импульс FSR, а в остальном она совпадает с рисунком 4.48. Если в процессе приема приходит импульс FSR (ошибка), то операция приема прекращается, один пакет данных теряется и начинается новый цикл приема.

4.4.8 Мультиплексируемый последовательный порт с разделением по времени

Мультиплексируемый последовательный порт с разделением по времени (TDM) позволяет последовательно подключать к процессору 1867ВЦ2Т (1867ВЦ2АТ) до семи других устройств. TDM порт, кроме того, обеспечивает простой и эффективный интерфейс для мультипроцессорных приложений.

Посредством бита TDM в управляющем регистре последовательного TDM порта TSPC, порт может быть сконфигурирован в мультипроцессорном режиме (TDM=1) или в автономном режиме (TDM=0). В этой главе описаны операции TDM порта, когда он сконфигурирован в мультипроцессорном режиме. Порт может быть закрыт для пониженного потребления мощности через XRST и RRST биты.

4.4.8.1 Мультиплексирование по времени

Осноаной режим мультиплексирования с разделением по времени – это разделение временных интервалов на несколько подинтервалов, каждый из которых представляет собой коммуникационный канал в соответствии с предварительно определенным

порядком. Рисунок 4.51 показывает 4-х канальную схему TDM. Первый временной слот обозначается как канал 1 (chan 1), второй – как канал 2 (chan 2) и т.д. Канал 1 активен в течение первого периода коммуникации и затем в течение каждого следующего четвертого периода. Оставшиеся три канала чередуются по времени за каналом 1.

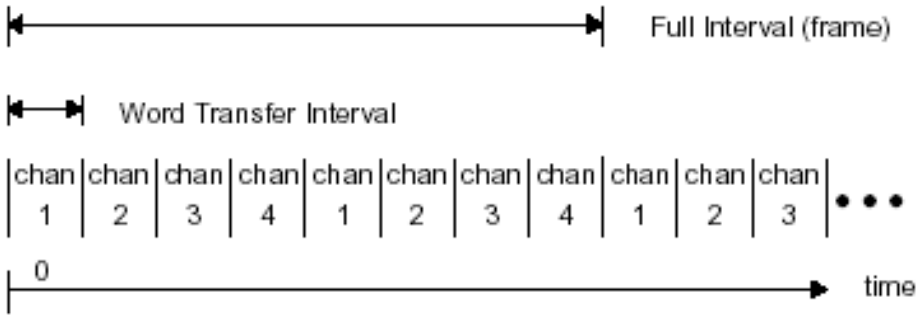


Рисунок 4.51 - Мультиплексирование с разделением по времени

TDM порт процессора 1867ВЦ2Т (1867ВЦ2АТ) использует 8 TDM-каналов. Все элементы, передаваемые или получаемые по каждому из каналов, могут быть заданы независимо. Это позволяет достигать высокой степени гибкости в межпроцессорных связях.

4.4.8.2 Регистры интерфейса последовательного TDM порта

Последовательный TDM порт доступен через 6 картированных в памяти регистров и два других регистра (TRSR и TXSR), к которым нет прямого доступа в программе, но которые используются в реализации возможности двойной буферизации. Эти 8 регистров представлены в таблице 4.30.

Таблица 4.30 - Регистры последовательного TDM порта

Адрес	Регистр	Описание
0030h	TRCV	TDM регистр приема данных
0031h	TDXR	TDM регистр передачи данных
0032h	TSPC	TDM регистр управления последовательного порта
0033h	TCSR	TDM регистр выбора канала
0034h	TRTA	TDM регистр адреса приема /передачи
0035h	TRAD	TDM регистр адреса приема
——	TRSR	TDM сдвиговый регистр приема данных
——	TXSR	TDM сдвиговый регистр передачи данных

− TDM – регистр приема данных (TRCV). 16-битный TDM – регистр приема данных содержит входящие данные TDM – порта. TRCV имеет ту же самую функцию, что и DRR в последовательном порту (SP).

− TDM – регистр передачи данных (TDXR). 16-битный TDM – регистр передачи данных содержит передаваемые данные TDM – порта. TDXR имеет ту же самую функцию, что и DXR.

− Контрольный регистр последовательного TDM порта (TSPC). 16-битный TSPC содержит управление режима и бит состояния интерфейса последовательного TDM порта. TSPC идентичен SPC в последовательном порту (SP), за исключением того, что нулевой бит является битом управления TDM режимома. Бит TDM конфигурирует порт в режиме TDM (TDM=1) или в автономном режиме (TDM=0). В автономном режиме порт работает как стандартный последовательный порт(SP).

− TDM – регистр выбора канала (TCSR). 16-битный TCSR определяет, в какой временной слот(ы), должен передавать каждый из процессоров 1867ВЦ2Т (1867ВЦ2АТ).

− TDM – регистр приема/передачи адреса (TRTA) определяет в восьми младших битах (RA0 – RA7) адрес приема процессора 1867ВЦ2Т (1867ВЦ2АТ), а в восьми старших битах (TA0 – TA7) –адрес передачи.

− TDM – регистр адреса приема данных (TRAD). 16-битный TRAD содержит различную информацию относительно состояния TDM адресной линии (TADD).

− Сдвиговый регистр приема данных TDM (TRSR). 16-битный TRSR позволяет хранение данных, поступающих со входа приема данных (TDR), и передачи этих данных в TRCV в последовательном порту (SP).

− Сдвиговый регистр передаваемых данных TDM (TXSR). 16-битный TXSR управляет перемещением выходных данных из TDXR и хранит данные, для передачи их на вывод TDX. TXSR выполняет такую же функцию, как XSR.

4.4.8.3 Операции поседовательного TDM порта

Рисунок 4.52 показывает архитектуру TDM порта применительно к процессору 1867ВЦ2Т (1867ВЦ2АТ). На четырех-разрядной последовательной шине могут располагаться до восьми устройств. Эта четырех-разрядная последовательная шина состоит из трех шин последовательного порта: тактовых сигналов, кадровой

синхронизации и данных (TCLK, TFRM, TDAT) плюс дополнительная шина (TADD), которая содержит адресную информацию устройства. Следует отметить, что TDAT и TADD – двунаправленные шины и часто управляются разными устройствами на шине в течение различных промежутков времени внутри данного кадра операции.

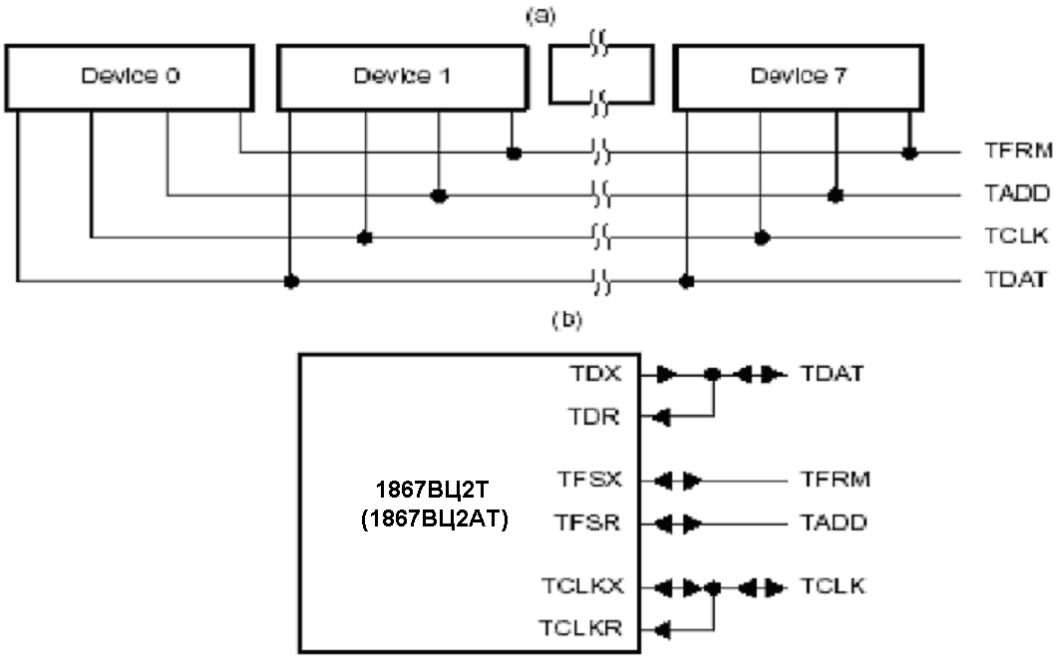


Рисунок 4.52 - Четырех-разрядная TDM шина

Линия TADD, которая управляется конкретным устройством, в конкретный момент времени определяет, какое устройство TDM конфигурации должно выполнять TDM – прием в течение этого промежутка (слота) времени. Это подобно допустимой операции чтения последовательного порта за исключением того, что некоторые соответствующие TDM регистры называются по-другому. TDM-регистр приема данных – это TRCV, сдвиговый TDM-регистр приема данных – это TRSR. Данные передаются через двунаправленную линию TDAT.

Обратите внимание, что на рисунке 4.52 (b) выводы TDX и TDR объединены вместе, для формирования линии TDAT. Кроме этого, только одно устройство может управлять линией данных и адреса (TDAT и TADD) в конкретном слоте. Выводы TDAT и TADD всех остальных устройств должны находиться в третьем состоянии в течении этого срока, что достигается соответствующим программированием регистра управления TDM порта. Между тем, в этом конкретном слоте все устройства (включая и то, которое управляет этим слотом) производят выборку линий TDAT и TADD, чтобы определить, представляет ли текущая передача достоверные данные, которые будут прочитаны одним

из устройств,подключенным к шине. Когда устройство распознает адрес, который оно считает соответствующим допустимым, тогда и происходит процесс TDM – чтения и данные передаются из TRSR в TRCV регистр. Генерируется прерывание по приему (TRNT), которое показывает, что TRCV содержит достоверные принятые данные, и может быть прочитан.

Все операции TDM порта синхронизируются сигналами TCLK и TFRM. Каждый из них генерируется только одним устройством, называемым TCLK и TFRM источником. Следовательно, все другие устройства в TDM конфигурации используют эти сигналы как входы. На рисунке 4.52 (b) выводы TCLKX и TCLKR внешне объединены вместе, для формирования линии TCLK. TFRM и TADD образуются из выводов TFSX и TFSR соответственно. Это сделано для того, чтобы облегчить использование TDM порта в стандартном режиме.

Операции TDM порта управляются шестью картированными в памяти регистрами. Расположение этих регистров показано на рисунке 4.53. Регистры TRCV и TDXR выполняют такие же функции, что и регистры DRR и DXR соответственно. Регистр TSPC идентичен регистру SPC за исключением того, что нулевой бит служит для управления TDM режимом. Этот бит конфигурирует порт в TDM режим (TDM=1) или в автономный режим (TDM=0). В автономном режиме порт работает как стандартный последовательный порт.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRCV	Receive Data															
TDXR	Transmit Data															
TSPC	Free	Soft	X	X	XRDY	RRDY	IN1	IN0	$\overline{\text{RRST}}$	$\overline{\text{XRST}}$	TXM	MCM	X	0	0	TDM
TCSR	X	X	X	X	X	X	X	X	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
TRTA	TA7	TA6	TA5	TA4	TA3	TA2	TA1	TA0	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
TRAD	X	X	X2	X1	X0	S2	S1	S0	A7	A6	A5	A4	A3	A2	A1	A0

Рисунок 4.53 - Диаграмма регистров последовательного TDM порта

При выборе TDM режима , биты ДDLB и FO в TSPC аппаратно конфигурируются в 0, в результате чего становится недоступным режим цифровой обратной связи и фиксируется длина слова в 16 бит. Также значения FSM бита не влияют на работу порта, когда TDM=1 и состояние флагов (биты RSRFULL, $\overline{XRESEMPTY}$ и т. п.) не определены.

Если TDM=1, изменения, сделанные в содержимом TSPC, становятся эффективными при завершении канала 7 текущего кадра. Таким образом, данные TSPC не могут быть изменены для текущего каедра, изменения вступят в силу в следующем кадре.

Устройство источника для синхронных сигналов TCLK и TFRM устанавливается посредством MCM и TXM битов соответственно. Устройство источника TCLK определяется установкой бита MCM TSPC регистра в 1. Вывод TCLKX конфигурируется как вход, если MCM=0 и как выход, если MCM=1. В последнем случае (внутренний тактовый сигнал процессора) устройство, у которого MCM=1, обеспечивает тактовый сигнал (частота TCLK = ¼ частоты CLKOUT1) для всех устройств на шине TDM. Тактовый сигнал может быть подведен внешним источником, если MCM=0 для всех устройств. TFRM может также быть внешним, если TXM=0. Внешний TFRM должен соответствовать требованиям TDM синхронизации по отношению к TCLK для соответствующей операции. Не более чем одно устройство должно иметь MCM или TXM, установленными в 1 в произвольном периоде времени. Выбор устройства, которое будет являться источником тактовых сигналов и кадровой синхронизации, выполняется только один раз, во время системной инициализации.

TDM регистр выбора канала (TCSR) данного устройства определяет в каком временном слоте это устройство передает данные. 1 в одном или более битах 0-7 TCSR активизирует передатчик в течении соответствующего временного слота. Кроме того существует ограничение системного уровня: не более, чем одно устройство может совершать передачу во время одного и того же временного слота; устройства не проверяют, есть ли конфликтная ситуация при обращении к шине , слоты должны быть последовательно заданы. Как и в операции TSPC, запись в TCSR в течении текущего фрейма будет эффективна только в следующем фрейме. Данное устройство может совершать передачу более чем в один слот.

TDM регистр приема/передачи (TRTA) данного устройства, определяет две ключевые части информации. Младшие 8 бит (RA7-RA0) - адрес приема устройства, старшие 8 бит (TA7-TA0) TRTA - адрес передачи (рисунок 4.53). Адрес приема имеет 8-битовое значение, которое устройство сравнивает с 8-битовым значением на линии TADD в каждом слоте. Адрес приема устанавливает слоты, в которых это устройство может принимать в зависимости от адреса, представленного в этих слотах, как определено передающими устройствами. Этот процесс происходит в каждом устройстве в течении каждого слота.

Адрес передачи (TA7-TA0)– это адрес, посредством которого устройство управляет линией TADD во время операции передачи в указанный слот. Адрес передачи указывает, какие приемные устройства могут осуществлять TDM прием данных.

Только одно устройство одновременно может управлять адресом передачи на TADD. Каждый процессор проводит побитную операцию логического суммирования своего адреса приема (RA7-RA0) со значением на линии TADD. Если эта операция приводит к значению, отличному от нуля, то выполняется достоверный TDM прием на процессоре или процессорах, чьи адреса совпадают с адресом передачи. Таким образом, при передаче одним устройством другим, один бит в верхней половине TRTA передающего устройства, имеющий значение «1», должен совпадать со знвчением бита в нижней половине TRTA (RA7-RA0) приемного устройства. Этот метод конфигурации TRTA позволяет одному устройству передавать данные другому устройству или устройствам, и любому одному устройству получать данные из одного или более устройств.

TDM регистр адреса приема (TRAD) содержит различную информацию о состоянии TADD линии, которая может опрашиваться для контроля предыдущих значений сигнала и взаимодействий между циклами команд и синхронизацией TDM порта.

Разряды 13-11 (X2-X0) содержат значение номера текущего слота, независимо от того был ли выполнен прием данных в этом слоте или нет. Эти данные фиксируются в начале слота и сохраняются до конца этого слота.

Разряды 8-10 (S2-S0) содержат номер последнего слота плюс 1 (по модулю), в котором данные были получены, например если последнее считывание данных имело место в пятом слоте предыдущего фрейма, эти биты будут содержать число 6). Эти значения фиксируются в течении прерывания по TDM приему в конце слота, в котором были получены последние достоверные данные, и сохраняются до окончания следующего слота, в котором происходит достоверный прием данных.

Разряды A7-A0 хранят последний адрес, выбранный на TADD линии, независимо от того, выполнена ли операция приема достоверных или нет.

4.4.8.4 Операции передачи и приема данных в TDM режиме

На рисунке 4.54 представлена временная диаграмма TDM порта при передаче. Сигналы TCLK и TFRM формируются источником синхронизации. Частота TCLK равна ¼ частоты CLKOUT1, если генерируется процессором 1867ВЦ2Т (1867ВЦ2АТ). TFRM

импульс формируется через каждые 128 циклов TCLK и синхронизирован для совпаденя с 0 битом 7 слота, который является последним битом предыдущего фрейма. Взаимосвязь TFRM и TCLK позволяет управлять 16 битами данных для каждого из 8 временных слотов на TDAT линии, которая также дает возможность процессору выполнять максимум из 64 инструкций в течении каждого слота, предполагая, что используется внутренняя синхронизация процессора 1867ВЦ2Т (1867ВЦ2АТ). Начиная с 0 слота и считая MSB первым, передающее устройство управляет 16 битами данных для каждого слота, где каждый разряд имеет длительность, равную циклу TCLK, исключая первый бит каждого слота, длительность которого составляет половину продолжительности одного бита. Следут заметить, что данные на TDAT линии синхронизируются передающим устройством и выбираются из TDAT линии приемным устройством на переднем фронте TCLK.

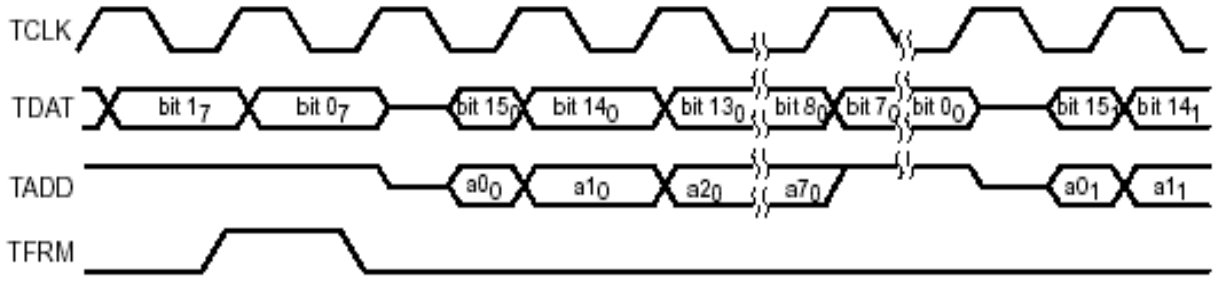


Рисунок 4.54 - Синхронизация последовательного порта (состояние TDM)

Одновременно с передачей данных передающее устройство так же управляет TADD линией с адресом передачи для каждого слота. Эта информация, в отличие от TDAT, имеет размер только в один байт и передается, считая LSB первым, для первой половины каждого слота. В течении второй половины слота (последние восемь периодов TCLK) TADD линия устанавливается в 1. Логика TDM приема выбирает TADD линию только в течении первых восьми периодов TCLK, игнорируя ее в течении второй половины слота. Поэтому передающее устройство (если это не 1867ВЦ2Т, 1867ВЦ2АТ) может установить TADD в 1 или 0 в течении этого периода времени. Следует заметить, что, также как и в TDAT, первый бит TADD передается последним только для одной половины одного TCLK цикла.

Если нет устройства на шине TDM, сконфигурированого для передачи в слот (т.е. ни одно из устройств не имеет 1 для соответствующего слота в регистре TCSR), то слот считается пустым. В пустом слоте линии TADD и TDAT имеют третье состояние. Для исключения ложного чтения устройством 1 кОм-ный резистор с подкачкой 0 должен быть

связан с TADD линией. Это является причиной того, что TADD линия при чтении начодится в 0 в пустом слоте. В противном случая, любой случайный сигнал на линии TADD, который возникает при сравнении крнкретного адреса получения, будет вызывать ложное чтение. Если рассеяние мочности учитывается и резистор не желателен, то произвольный процессор с адресом передачи, равным 0h, может управлять пустыми слотами записью в TDXR в этих слотах. Резистор в 1 КОм на TDAT линии не нужен.

Пустой слот TDM может образовываться в следующих случаях: первый случай, как уже упомянуто выше, имеет место, когда ни одно из устройств не имеет сконфигурированного TCSR для передачи в этотом слоте. Второй случай: TDXR не загружен перед слотом передачи в отдельном фрейме. Это также может произойти, когда содержимое TCSR изменяется, но фактическое содержимое TCSR не обновляется, пока не появится следующий импульс TFRM. Поэтому любые последующие изменения будут эффективны только в следующем фрейме. Это справедливо для адреса приема (младшие 8 разрядов TRTA). Адрес передачи (старшие 8 разрядов TRTA) и TDXR могут быть изменены внутри текущего фрейма для отдельного слота, предполагая, что слот еще не был достигнут когда выполняется инструкция для загрузки TRTA или TDXR. Следует заметить, что для загрузки ажреса передачи нет необходимости каждый раз загружать TDXR; когда произошла запись в TDXR и началась передача, текущий адрес передачи передается на TADD линию.

Номер текущего слота может быть получен при чтении X2-X0 битов регистра TRAD. Это позволяет сделать гибкой конфигурацию TDM порта в базисе «слот за слотом» и даже возможно совместное использование слотов. Ключом для использования этих возможностей является понимание вреиенных соотношений между выполнением инструкций и фреймом/слотами TDM порта.

Если TDM порт будет переконфигурироваться от «слота к слоту», то изменения в соответствующих регистрах должны делаться достаточно быстро, чтобы желаемый эффект достигался в желаемое время.

Необходимо принять во внимание, что изменение регистра TCSR и адреса приема (младшая половина TRTA) произойдет только после начала нового фрейма. В то время, как изменение адреса передачи (старшая половина TRTA) и TDXR (передаваемые данные) пройдут в начале следующего слота, как было сказано выше.

4.4.8.5 Условия исключения интерфейса последовательного TDM порта

Всвязи с особенностями TDM порта, когда один процессор может передавать в различных слотах, понятия переполнение и недозагрузка становятся неактуальными. Поэтому флаги переполнения и недозагрузки не активны в TDM режиме. В приемном TDM порту, если TRCV небыл прочитан и порт готовится принять еще одно слово (когда адрес, передаваемый на линии TADD, совпал с адресом приема в регистре TRTA), то в этом случае перезаписывается то значение, которое в нем было; приемный порт не останавливается. С другой стороны, если TDXR не был обнавлен перед передачей, то линии TADD или TDAT не управляются и эти выводы остаются в третьем состоянии. Этот режим предотвращает возможность ложной передачи.

Если импульс TFRM возникает не вовремя в течении фрейма, то TDM порт не способен продолжать правельно работать и номера битов и слов становятся неопределенными, когда это случается. Только один импульс TFRM должен возникнуть каждые 128 тактов TCLK. В отличии от последовательного порта, TDM порт не может переинициализироваться импульсом синхронизации фрейма в процессе передачи. Для корректировки неправельно поданного импульса TFRM TDM порт должен быть сброшен.

4.4.8.6 Примеры работы интерфейса TDM порта

В этом примере восемь устройств соединены между собой через TDM порты, как покозано на рисунке 4.52.

Таблица показывет значение TADD для каждого из восьми каналов и обозначение передающих и приемных устройств. Пример показывает конфигурацию соединения устройств друг с другом. В этом примере Устройство0 передает другим семи устройствам адреса в течении слота 0. В последующих фреймах каждое из устройств.1-7 соединяются с другим устройством.

Таблица 4.31 - Сценарий межпроцессорных связей.

Каналы	Данные TADD	Передающее устройство	Приемные устройства
0	0FFh	0	1-7
1	40h	7	6
2	20h	6	5
3	10h	5	4
4	08h	4	3

Продолжение таблицы 4.31

1	2	3	4
5	04h	3	2
6	02h	2	1
7	01h	1	0

Таблица 4.32 показывает содержимое регистров TDM портов каждого устройства, каждого устройства, которое соответствует сценарию, представленному в таблице А1.

Устройство 0 вырабатывает TCLK и TFRM сигналы для всех каналов и устройств. Содержимое TSCR и TRTA определяют, какое из устройств должно передавать в данном канале и какое из устройств должно принимать.

Таблица 4.32 - Содержание регистров TDM порта.

Устройства	TSPC	TRTA	TCSR
0	xxF9h	0FE01h	xx01h
1	xxC9h	0102h	xx80h
2	xxC9h	0204h	xx40h
3	xxC9h	0408h	xx20h
4	xxC9h	0810h	xx10h
5	xxC9h	1020h	xx08h
6	xxC9h	2040h	xx04h
7	xxC9h	4080h	xx02h

В этом примере адреса передачи передающих устройств (старшие биты TRTA) сравниваются с адресами приема (младшие биты TRTA) приемных устройств. Следует заметить, что не обязательно полное совпадение адресов приема и передачи; операция сравнения в приемном порту представлена как побитная операция AND. Поэтому необходимо, чтобы совпал только один бит из сравниваемых адресов.

Устройство также может передавать самона себя, т. к. обе операции приема и передачи происходят по переднему фронту TCLK. Для того, чтобы обеспечить этот режим необходимо использовать стандартное соединение интерфейса TDMпорта, как показано на рисунке 4.52 . Если в регистре TRTA устройства 0 записано 0101h, то оно будет передавать только само на себя.

5 Проработка вопросов отладки, отладочные средства для ИМС

Разрабатываемые микропроцессоры в соответствии с ТЗ являются полным функциональным аналогом высокопроизводительного 16-разрядного микропроцессора 1867ВМ2, поэтому для него применимы аппаратно-программные средства фирмы Texas Instrument и других производителей, разработанные для аналога.

Для отладки систем на основе микроконтроллера 1867ВЦ2Т (1867ВЦ2АТ) разработана и поставляется обширная номенклатура программных и аппаратных средств, в том числе:

- компиляторы С, С++ с интегрированной средой под Windows-XX;
- ассемблеры, дизассемблеры;
- операционные системы реального времени;
- отладчики-симуляторы;
- внутрисхемные эмуляторы;
- наборы Starter Kit's.

5.1 Программные средства поддержки разработок, среда разработчика Code Composer Studio

Code Composer Studio - интегрированная среда разработчика, имеющая удобный графический интерфейс в сочетании с мощными средствами конфигурирования и отладки, ориентированными на ЦОС приложения. При ее разработке было достигнуто оптимальное сочетание визуальных средств конфигурирования и разработки с мощностью и возможностями продукта.

Система ориентирована на создание максимального удобства для разработчика. Ее применение позволяет в несколько раз сократить сроки разработки и отладки ЦОС систем.

Являясь полностью функционально законченным продуктом Code Composer Studio позволяет не выходя из отладочной среды редактировать, компилировать и отлаживать программы.

Code Composer Studio имеет комфортный графический оконный интерфейс, сделанный аналогично интерфейсу MS VisualC. Возможно открытие любого количества окон просмотра кода и дизассемблирования, одновременная отладка на С и ассемблере, одновременная работа с несколькими процессорами.

Code Composer Studio имеет гибкую систему настроек, а также С-подобный внутренний язык сценариев, который позволяет настроить интерфейс практически как угодно, вплоть до добавления пунктов в меню. Такая возможность написания собственных сценариев и встраивание их в интерфейс позволяет сделать рабочее место максимально удобным.

Мощные возможности анализа и отладки систем в реальном времени дают возможность отлаживать и анализировать поведение системы без остановки процессора.

Уникальные возможности визуализации данных и состояния системы позволяют быстрее и результативнее оценивать состояние устройства.

Для специфических приложений, таких как телекоммуникации, задачи управления и обработка изображений особенно важны возможности Code Composer Studio по графической визуализации данных в реальном времени.

Работа в Code Composer Studio, использующая новые технологии TI, такие как Real Time Data Exchage и DSP BIOS, позволяет существенно уменьшить время разработки и освобождает разработчика от решения текущих проблем, позволяя сосредоточиться непосредственно на разработке приложений.

Основные новшества Code Composer Studio, по сравнению с предыдущей версией, включают в себя:

- Непрерывный обмен между ЦСП и отладчиком в реальном времени.
- Анализ и отладка в реальном времени.
- Визуализация состояния системы в реальном времени.
- Расширенная визуализация, включающая быстрое преобразование Фурье, временной анализ, множество видов стандартных диаграмм и многое другое.
- Возможность подключения нескольких различных отладочных систем.
- Возможность работы с несколькими различными процессорами на одном JTAG канале.
- Сетевой менеджмент проектов, позволяющий координировать совместную разработку проектов группой.
- Готовность к отладке с использованием ОС реального времени.

5.2 Аппаратные средства поддержки разработок

Для аппаратной поддержки разработок на основе микропроцессоров ‘C50 предлагается широкий спектр средств, как фирмы Texas Instrument, так и других фирм, которые можно классифицировать как по назначению, так и по стоимости внутри каждого класса. Можно выделить следующие классы аппаратных средств по их назначению:

- стартовые наборы разработчика (Starter Kits);
- внутрисхемные эмуляторы (In-Circuit Emulators);
- специализированные наборы разработчика.

Некоторые аппаратные средства являются универсальными и одновременно могут быть отнесены к нескольким классам, например к стартовым наборам разработчика и внутрисхемным программаторам.

Внутрисхемныq эмулятор SDSP-510 с интерфейсом JTAG 3/5V предназначен для отладки аппаратного и программного обеспечения процессоров фирмы Texas Instrument серий C2xx/C5x/C54x.

Внутрисхемный эмулятор состоит из платы, устанавливаемой в слот ISA и кабеля для подключения к отлаживаемому устройству.

Отлаживаемый процессор и эмулятор соединяются по 5 проводному последовательному интерфейсу JTAG через специальные выделенные на процессоре выводы. Такое подключение дает возможность отлаживать устройство в той конфигурации и на том процессоре, с которым оно будет работать. Это снимает как вопросы быстродействия эмулятора, так и вопросы изменения электрических параметров при подключении эмулятора.

Подключение внутрисхемного эмулятора полностью прозрачно для исполняемой программы и не оказывает на ее выполнение никакого влияния. При этом программа исполняется на полной скорости процессора без каких-либо задержек и ограничений по производительности.

Внутрисхемный эмулятор позволяет:

- производить загрузку команд программы и данных как в ОЗУпроцессора, так и во внешнее ОЗУ;
- устанавливать любое количество точек останова;
- производить контроль и модификацию содержимого памяти, регистров процессора и регистров периферийных устройств;
- проводить пошаговое выполнение программы;
- измерять время выполнения программы или ее частей.

К одному внутрисхемному эмулятору SDSP-510 одновременно может подключаться несколько одновременно отлаживаемых процессоров.

Программное обеспечение не входит в состав внутрисхемного эмулятора и должно приобретаться отдельно для каждого из семейств процессоров.

6 Указания по применению и эксплуатации

6.1 Микросхемы должны использоваться в соответствии с указаниями по применению и эксплуатации микросхем по ОСТ В 11 0998-99 с дополнениями и уточнениями, приведенными в настоящем разделе.

6.2 Эксплуатация микросхемы производится в соответствии с техническим описанием КФДЛ.431299.011, в котором приведено изложение принципа работы, архитектура и система команд.

6.3 При монтаже микросхемы все выводы GND, а также входы постоянно находящиеся при эксплуатации в состоянии низкого уровня (TRST, CLKMD1, CLKMD2 и т. п.) подключить к общей шине. На все выводы U_{CC} относительно корпуса подключить бескорпусные ёмкости типа К-16 величиной 0,1 мкФ. Входы/выходы BR, CLKX, CLKR, TCLKX, TCLKR подключить через соответствующий резистор нагрузки (2,2…10) кОм к шине питания ядра. При эксплуатации входы и входы/выходы микросхемы должны быть подключены к выводам, обрамляющих микросхем. Не используемые входы и входы/выходы микросхемы, с учетом выше изложенного, рекомендуется подключить через резисторы подпитки 10 кОм к шине питания ядра, кроме входов TMS, TCK, TDI, имеющих внутрисхемную подпитку. Не используемые выходы могут быть свободными.

6.4 При включении микросхемы в рабочий режим рекомендуется в первую очередь включить источник по шине питания ядра, а затем по шине питания периферии. Допускается одновременное включение источников питания ядра и периферии, при большей фильтрующей емкости по шине питания периферии.

6.4.1 Подача входных напряжений и сигналов синхронизации на микросхему разрешается после достижения номинального напряжения U_{CC1} ядра после или одновременно с включением напряжения питания периферии U_{CC2} в соответствии с системой команд и временными диаграммами.

6.4.2 Порядок снятия напряжений с микросхемы обратный подаче. В начале снимаются сигналы синхронизации и входные напряжения, а затем напряжения питания периферии и ядра.

7 Типовые характеристики электропараметров ИМС 1867ВЦ2Т, 1867ВЦ2АТ

На приведённых ниже рисунках представлены электрические параметры процессоров 1867ВЦ2Т, 1867ВЦ2АТ.

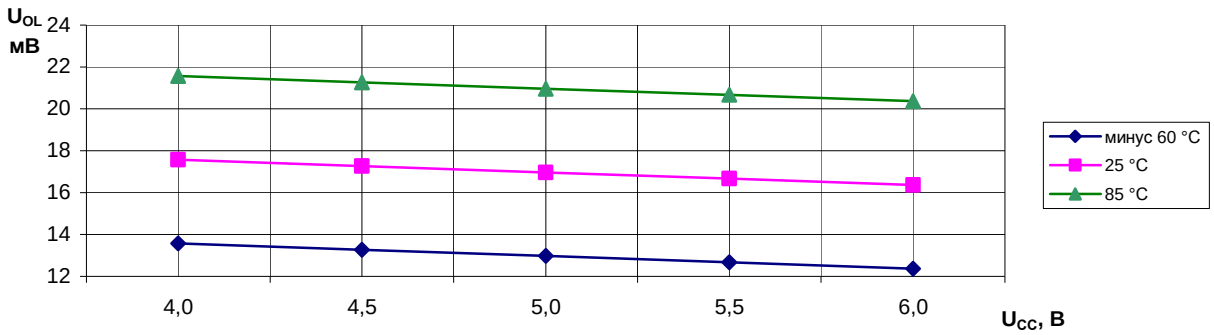


Рисунок 7.1 - Зависимость статического выходного напряжения низкого уровня от напряжения питания (U_{OL} = f(U_{CC}, T)) при I_{OL} = 2,0 мА

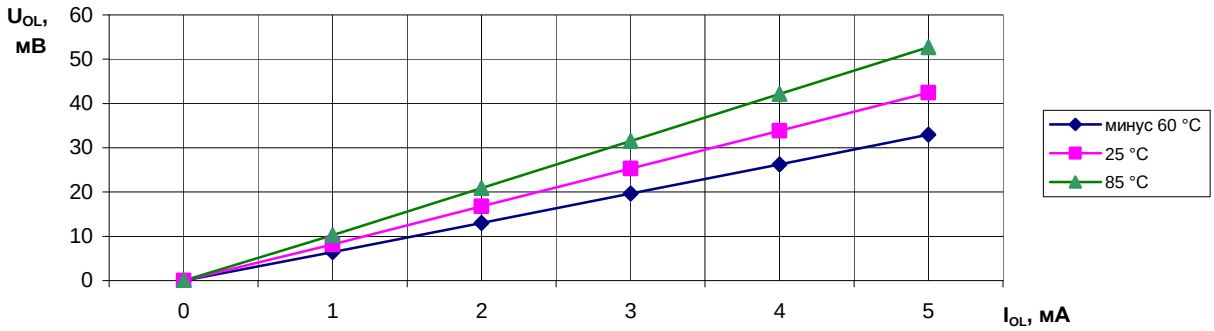


Рисунок 7.2 - Зависимость статического выходного напряжения низкого уровня от тока нагрузки (U_{OL} = f($I_{OL,}$ T)) при U_{CC} = 4,5 В

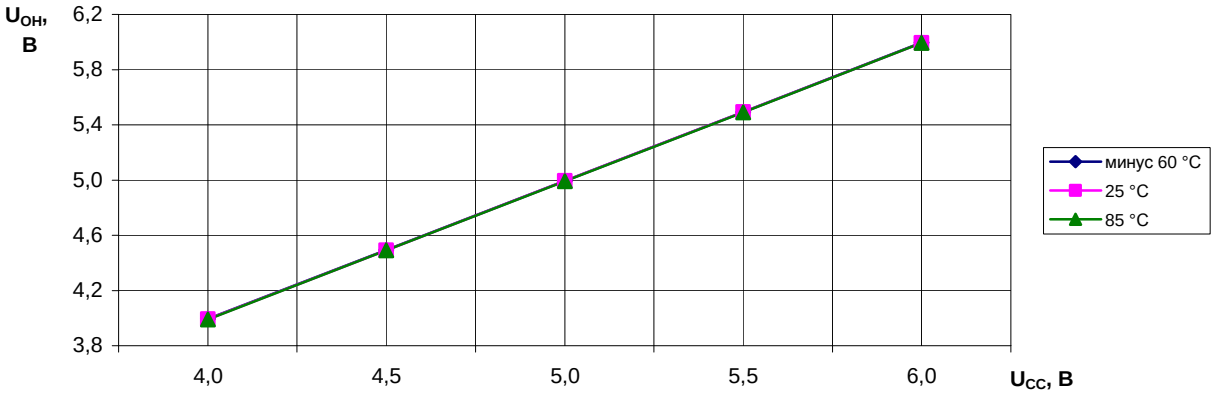


Рисунок 7.3 - Зависимость статического выходного напряжения высокого уровня от напряжения питания (U_{OH} = f(U_{CC}, T)) при I_{OH} = - 0,3 мА

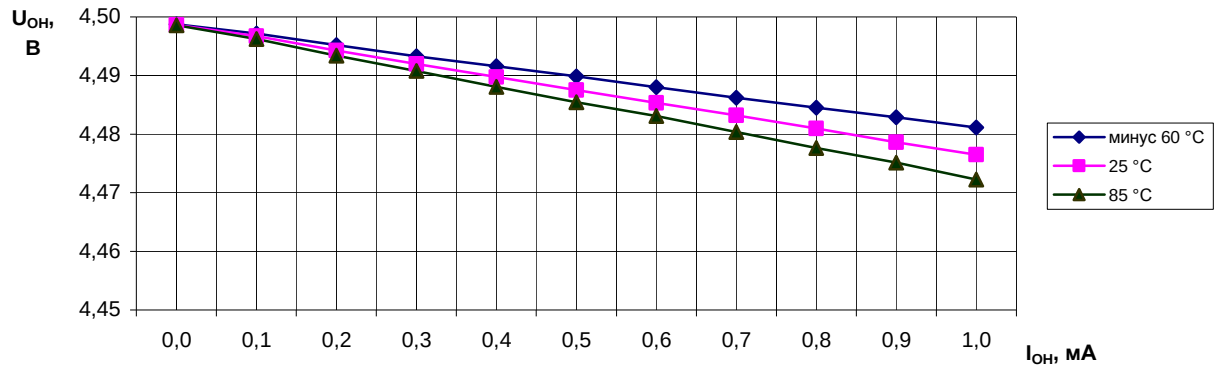


Рисунок 7.4 - Зависимость статического выходного напряжения высокого уровня от тока нагрузки ($U_{OH} = f(I_{OH}, T)$) при U_{CC} = 4,5 В

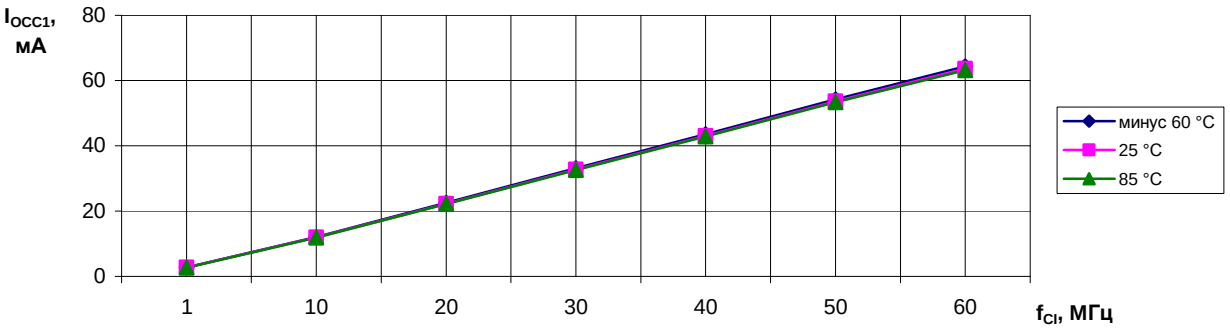


Рисунок 7.5а - Зависимость динамического тока потребления ядра от тактовой частоты ($I_{OCC1} = f(f_{CI}, T)$) на тесте ФК Start_X2 (CLKMD1=0; CLKMD2=0) с делением f_{CI} X2/CLKIN на 2 ($T_{CO} = 2T_{CI}$) при U_{CC} = 5,5 В, C_L = 50 пФ

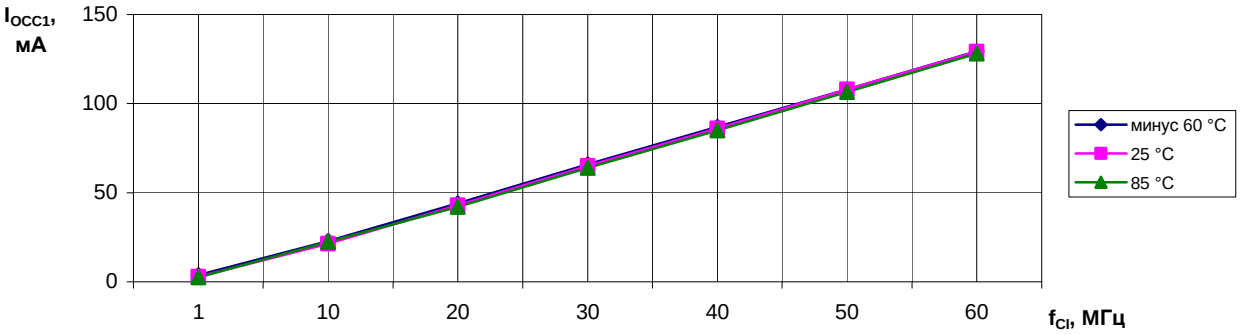


Рисунок 7.5б - Зависимость динамического тока потребления ядра от тактовой частоты ($I_{OCC1} = f(f_{CI}, T)$) на тесте ФК Start_IN2 (CLKMD1=1; CLKMD2=0) с делением f_{CI} CLKIN2 на 1 ($T_{CO}=T_{CI}$) при U_{CC} = 5,5 В, C_L = 50 пФ

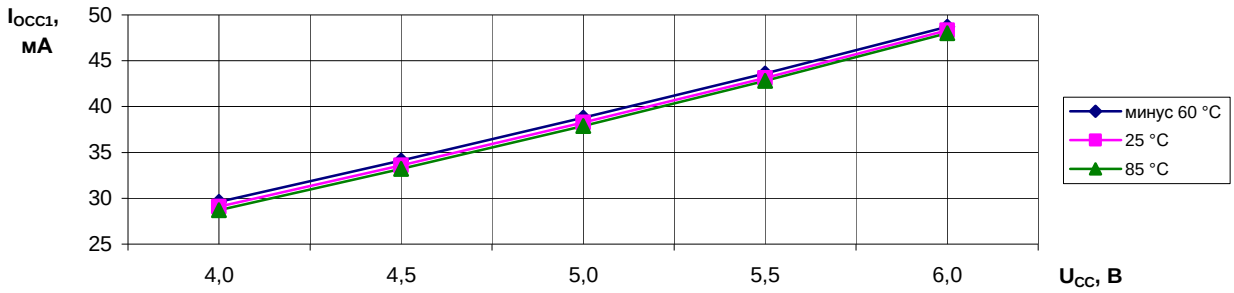


Рисунок 7.6а - Зависимость динамического тока потребления ядра от напряжения питания (I_{OCC1} = f(U_{CC}, T)) на тесте ФК Start_X2 (CLKMD1=0; CLKMD2=0) с делением f_{CI} X2/CLKIN на 2 (T_{CO} =2T_{CI}) при f_{CI} = 40 МГц, C_L = 50 пФ

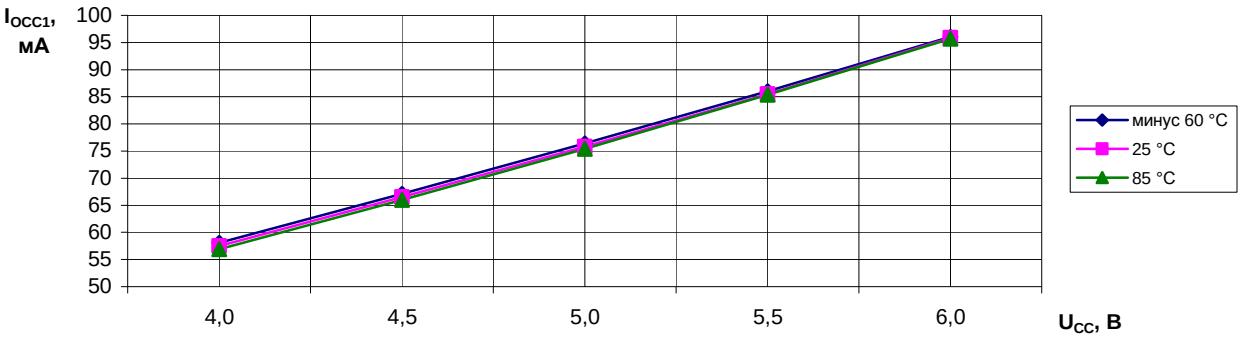


Рисунок 7.6б - Зависимость динамического тока потребления ядра от напряжения питания (I_{OCC1} = f(U_{CC}, T)) на тесте ФК Start_IN2 (CLKMD1=1; CLKMD2=0) с делением f_{CI} CLKIN2 на 1 (T_{CO} = T_{CI}) при f_{CI} = 40 МГц, C_L = 50 пФ

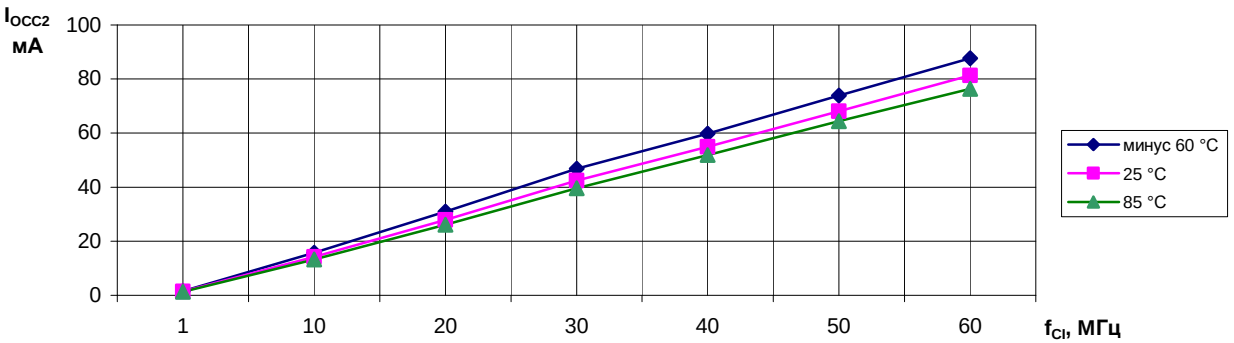


Рисунок 7.7а - Зависимость динамического тока потребления периферии от тактовой частоты (I_{OCC2} = f(f_{CI}, T)) на тесте ФК Start_X2 (CLKMD1=0; CLKMD2=0) с делением f_{CI} X2/CLKIN на 2 (T_{CO}=2T_{CI}) при U_{CC} = 5,5 В, C_L = 50 пФ

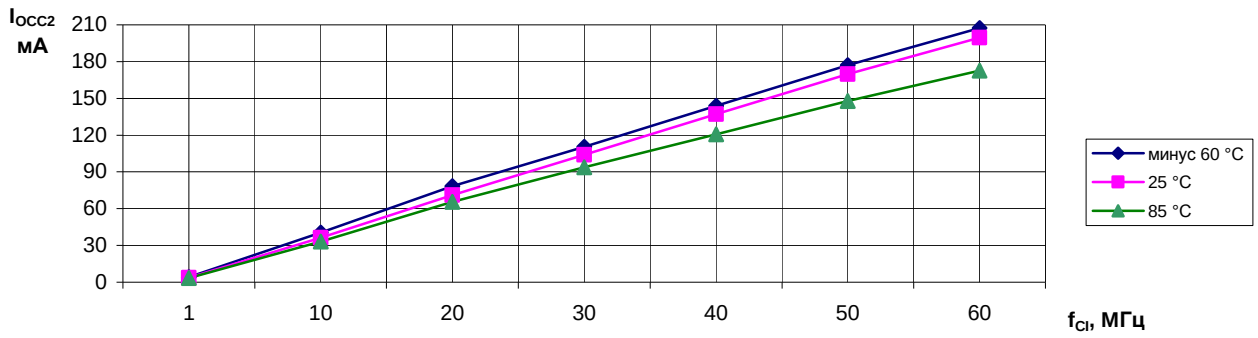


Рисунок 7.7б - Зависимость динамического тока потребления периферии от тактовой частоты (I_{OCC2} = f(f_{CI}, T)) на тесте ФК Start_IN2 (CLKMD1=1; CLKMD2=0) с делением f_{CI} CLKIN2 на 1 (T_{CO}=T_{CI}) при U_{CC} = 5,5 В, C_L = 50 пФ

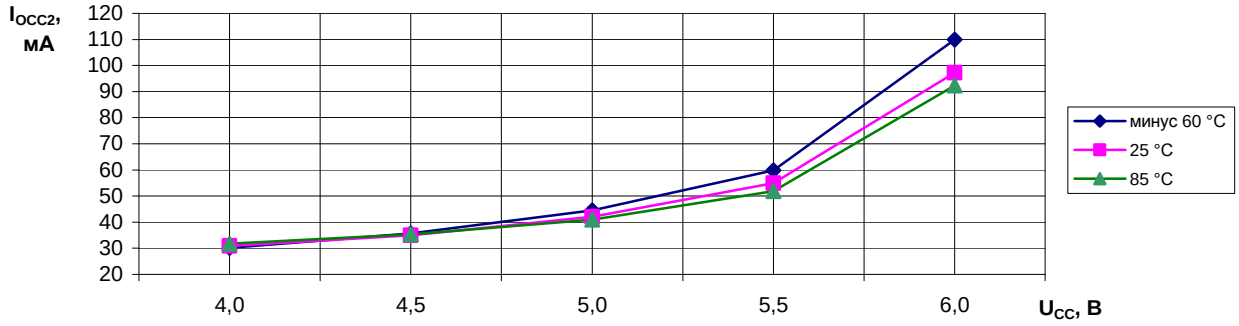


Рисунок 7.8а - Зависимость динамического тока потребления периферии от напряжения питания (I_{OCC2} = f(U_{CC}, T)) на тесте ФК Start_X2 (CLKMD1=0; CLKMD2=0) с делением f_{CI} X2/CLKIN на 2 (T_{CO} = 2T_{CI}) при f_{CI} = 40 МГц, C_L = 50 пФ

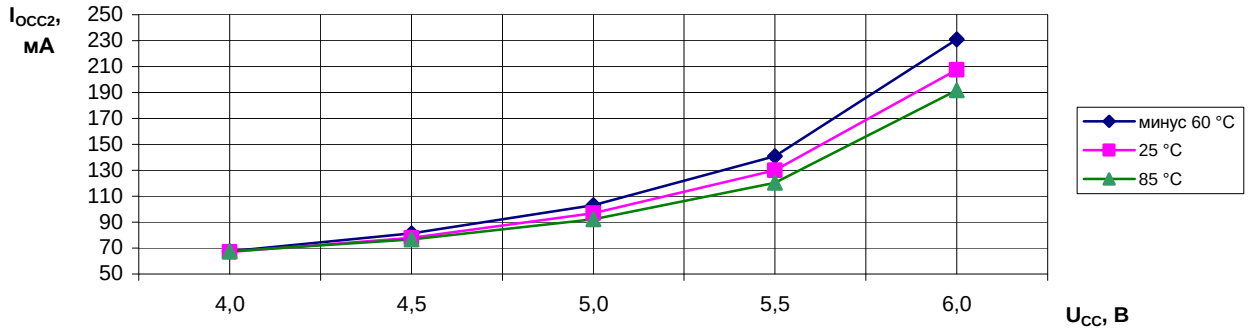


Рисунок 7.8б - Зависимость динамического тока потребления периферии от напряжения питания (I_{OCC2} = f(U_{CC}, T)) на тесте ФК Start_IN2 (CLKMD1=1; CLKMD2=0) с делением f_{CI} CLKIN2 на 1 (T_{CO} = T_{CI}) при f_{CI} = 40 МГц, C_L = 50 пФ

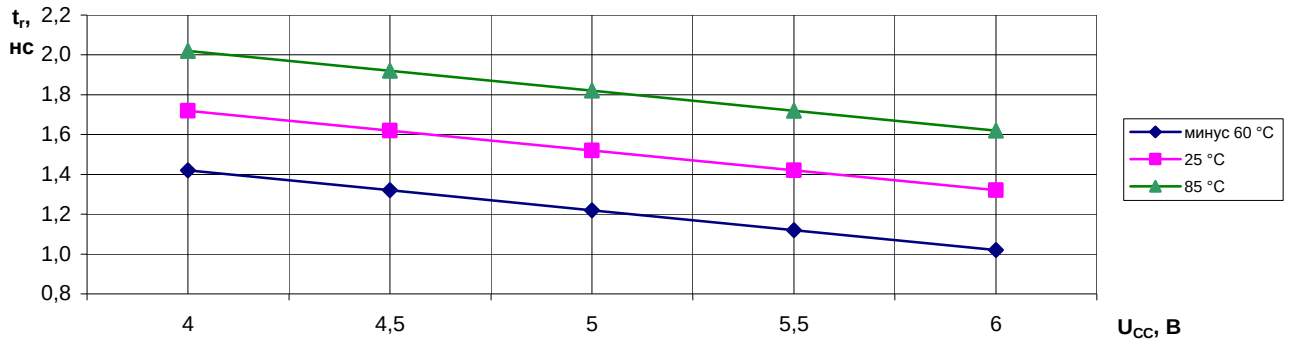


Рисунок 7.9 - Зависимость времени переключения на выходе CLKOUTI от напряжения питания ($t_r = f(U_{CC,} T)$) при C_L = 50 пФ, f_{CI} = 10 МГц

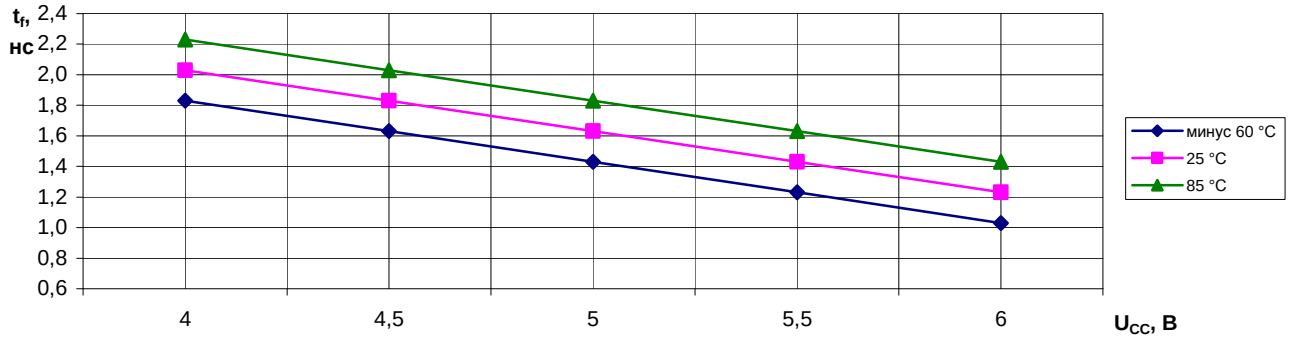


Рисунок 7.10 - Зависимость времени переключения на выходе CLKOUTI от напряжения питания ($t_f = f(U_{CC,} T)$) при C_L = 50 пФ, f_{CI} = 10 МГц

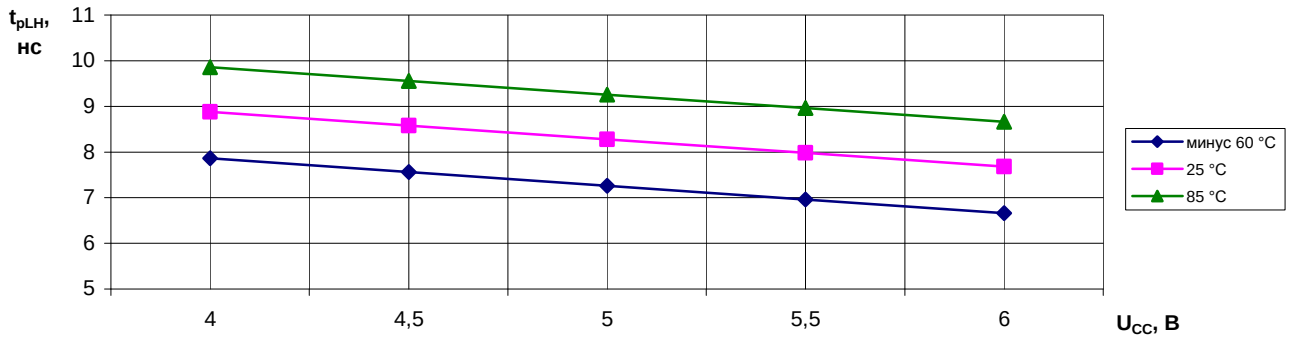


Рисунок 7.11 - Зависимость времени задержки распространения сигнала CLKOUTI относительно тактового сигнала CLKIN от напряжения питания ($t_{pLH} = f(U_{CC,} T)$) при C_L = 50 пФ, f_{CI} = 10 МГц

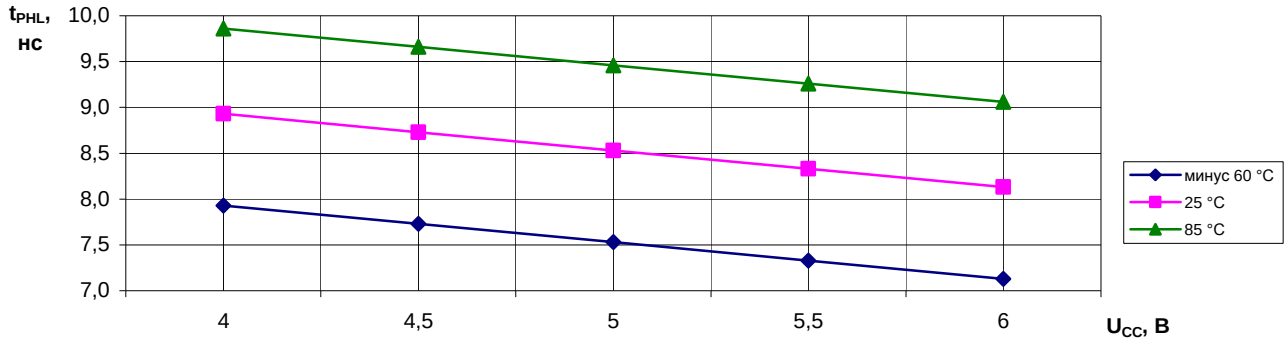


Рисунок 7.12 - Зависимость времени задержки распространения сигнала CLKOUTI относительно тактового сигнала CLKIN от напряжения питания ($t_{pHL} = f(U_{CC,} T)$) при C_L = 50 пФ, f_{CI} = 10 МГц

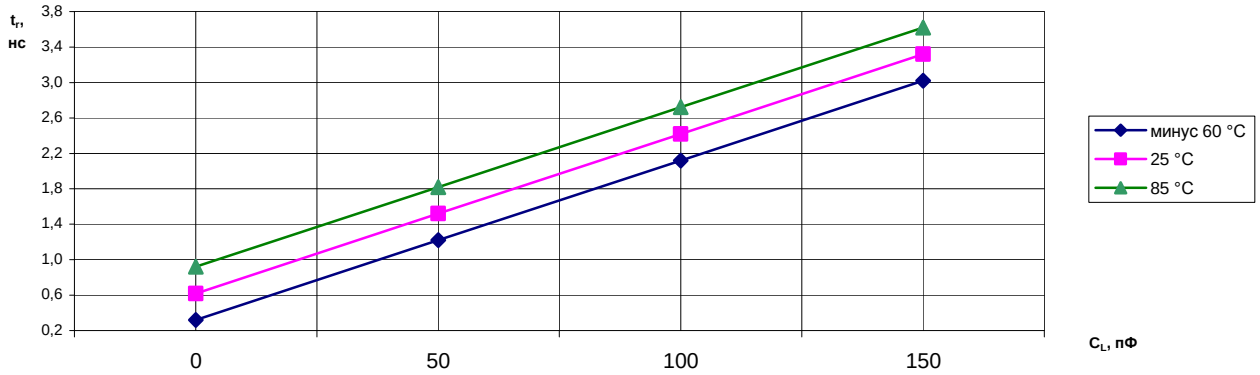


Рисунок 7.13 - Зависимость времени переключения на выходе CLKOUTI от ёмкостной нагрузки $t_r = f(C_L, T)$ при U_{CC} = 5,0 В, f_{CI} = 10 МГц

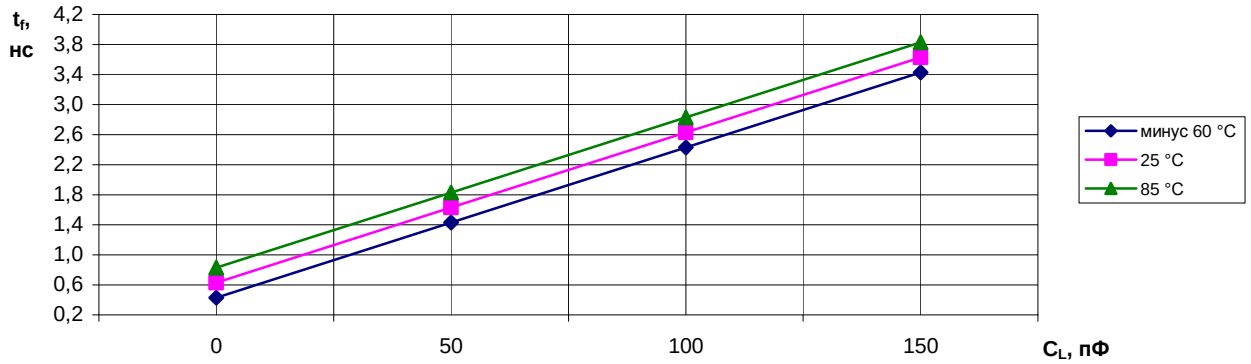


Рисунок 7.14 - Зависимость времени переключения на выходе CLKOUTI от ёмкостной нагрузки $t_f = f(C_L, T)$ при U_{CC} = 5,0 В, f_{CI} = 10 МГц

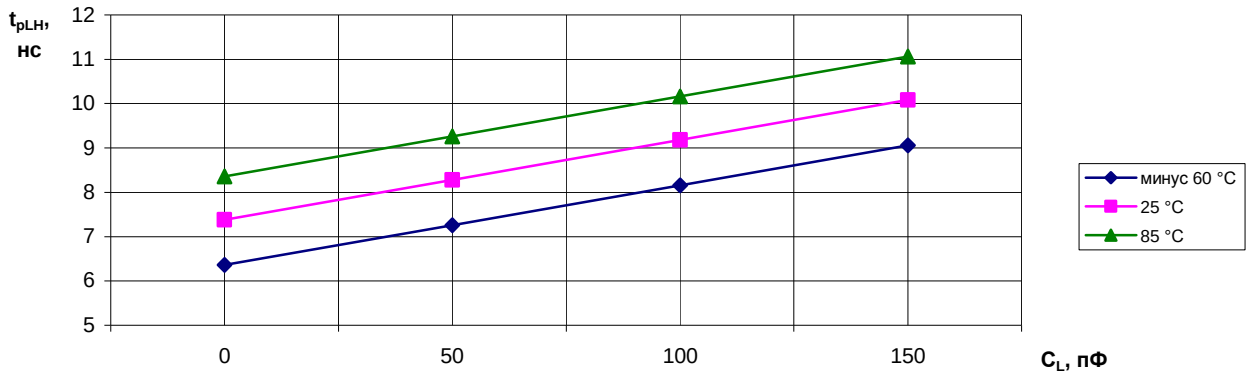


Рисунок 7.15 - Зависимость времени задержки распространения сигнала CLKOUTI относительно тактового сигнала CLKIN от ёмкостной нагрузки $t_{pLH} = f(C_L, T)$ при U_{CC} = 5,0 В, f_{CI} = 10 МГц

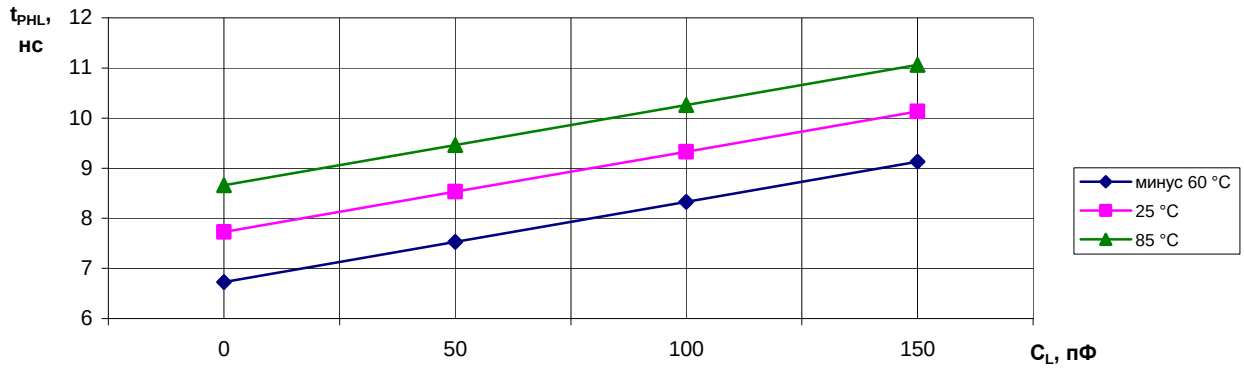


Рисунок 7.16 - Зависимость времени задержки распространения сигнала CLKOUTI относительно тактового сигнала CLKIN от ёмкостной нагрузки $t_{pHL} = f(C_L, T)$ при U_{CC} = 5,0 В, f_{CI} = 10 МГц

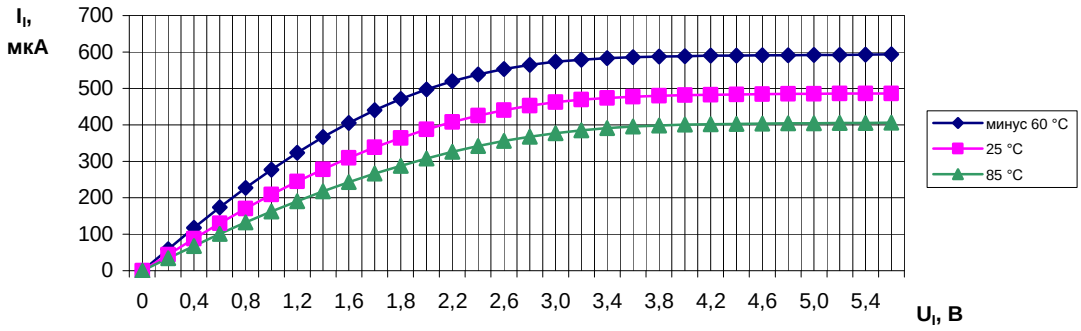


Рисунок 7.17 - Зависимость входного тока по входу TRST (подтянутого через резистор к GND) от входного напряжения ($I_I = f(U_I, T)$) при U_{CC} = 5,5 В

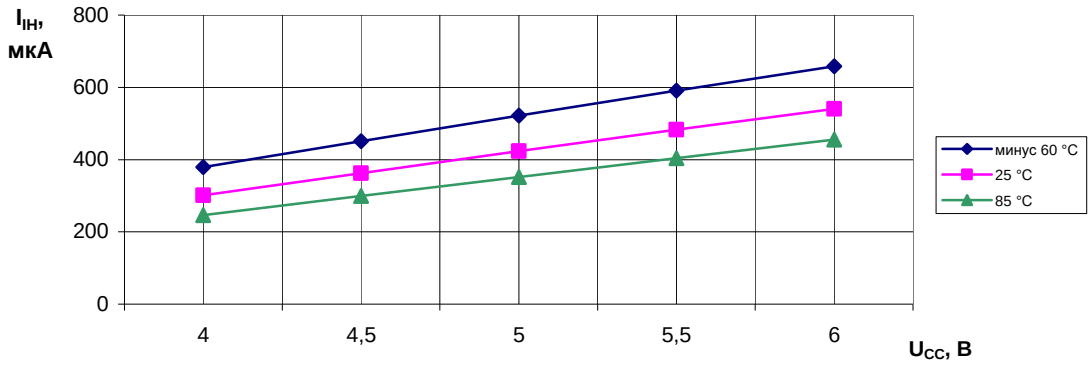


Рисунок 7.18 - Зависимость входного тока высокого уровня по входу TRST (подтянутого через резистор к GND) от напряжения питания ($I_{IH} = f(U_{CC}, T)$) при $U_{IH} = U_{CC}$

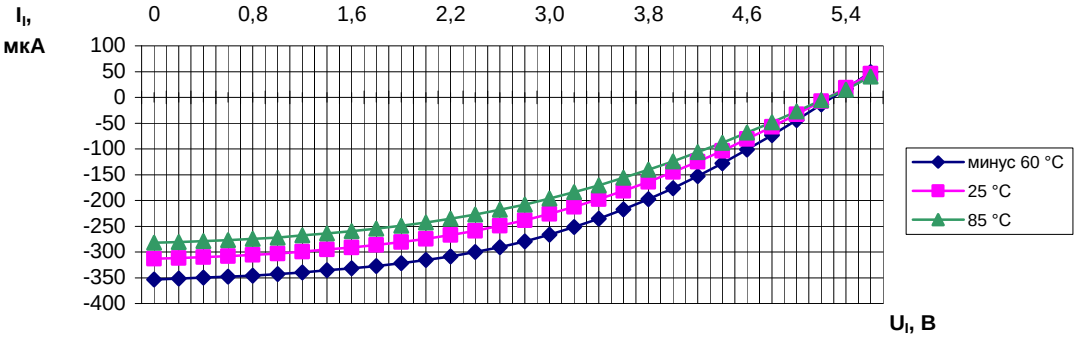


Рисунок 7.19 - Зависимость входного тока по входам TCK, TMS, TDI (подтянутых через резистор к U_{CC}) от входного напряжения ($I_I = f(U_I, T)$) при $U_{CC} = 5{,}5$ В

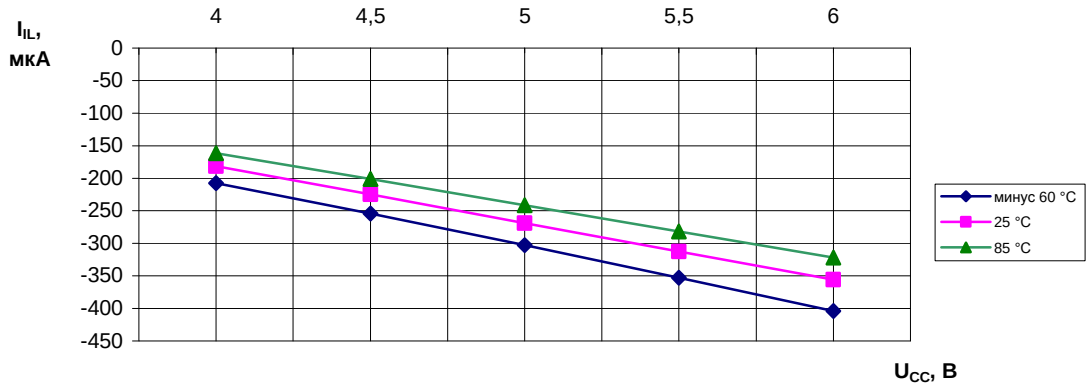


Рисунок 7.20 - Зависимость входного тока низкого уровня по входам TCK, TMS, TDI (подтянутых через резистор к U_{CC}) от напряжения питания ($I_{IL} = f(U_{CC}, T)$) при $U_{IL} = 0$ В

8 Расчет показателей надежности

Расчет проведен по методике РД 11 0755-90 «Микросхемы интегральные. Методы ускоренных испытаний на безотказность и долговечность» (метод 5-1) (далее РД).

1 Общая модель надежности ИС имеет вид:

$\lambda_{ИС}=K_{П}\times(\lambda_1+\lambda_2)$,

где:

- $\lambda_{ИС}$ – интенсивность отказов ИС при 25 ºС, (1/ч);
- K_n – коэффициент вида приемки, характеризующий систему отбраковочных испытаний. Для системы отбраковочных испытаний с ЭТТ K_n = 0,2 (таблица 7 РД);
- λ_1 – интенсивность отказов конструктивных элементов;
- λ_2 – интенсивность отказов элементов схемы.

2 Интенсивность отказов конструктивных элементов определяем по формуле:

$\lambda_1=\alpha_к\times\lambda_к+\alpha_{кр}\times\lambda_{кр}+n\times\alpha_c\times\lambda_c$,

где:

- $\lambda_к$ - интенсивность отказов корпуса. ИС выполнены в 132-выводных металлокерамических корпусах. Из таблицы 8 РД для 112-выводного (по числу задействованных выводов) корпуса $\lambda_к$ = 0,28 × 10^{-7} (1/ч);
- $\alpha_к$ - коэффициент, характеризующий различие корпусов разрабатываемых ИС и аналога и рассчитываемый по формуле: $\alpha_к = S_p/S_a$, где S_p , S_a – площадь поверхности разрабатываемой ИС и аналога. S_p=8,56 × 10^{-4} м^2, S_a =5,3× 10^{-4} м^2, $\alpha_к$ = 1,62. $\lambda_{кр}$ - интенсивность отказов соединений кристалла с основанием корпуса. Для способа крепления кристалла на клей $\lambda_{кр}$ = 0,01 × 10^{-7} (1/ч) (таблица 8 РД);
- $\alpha_{кр}$ - коэффициент, зависящий от площади кристалла. Для площади кристалла

$S_{кр}$ = 9,5 мм ×9,96 мм = 94,62 мм^2 $\alpha_{кр}$ = 3 (чертеж 2 РД);

- n - количество соединений выводов с кристаллом и корпусом. n = 224;
- λ_c - интенсивность отказов одного соединения. λ_c = 0,0015 × 10^{-7} (1/ч) для соединений Al-Al (таблица 8 РД);

- α_c - коэффициент, характеризующий конструктивные различия соединений разрабатываемых ИС и аналога, и определяемый по чертежу 3 РД в зависимости от площади сварных соединений S_c. Для $S_c = 0{,}7\times10^{-2}$ мм^2 $\alpha_c = 0{,}8$.

Подставив найденные значения в формулу для λ_1, получим:

$\lambda_1 = 1{,}62\times0{,}28\times10^{-7} + 3 \times 0{,}01 \times 10^{-7} + 0{,}8\times224 \times 0{,}0015 \times 10^{-7} = 0{,}752 \times 10^{-7}$ (1/ч).

3 Интенсивность отказов элементов схемы рассчитывается по формуле (51) РД с учетом влияния электрического режима и температуры. Эту формулу можно упростить, исходя из условия равномерной нагруженности различных участков схемы. Тогда:

$$\lambda_2 = \beta \times N \times \lambda_{эл} \times \alpha_I + \lambda_M \times S_M \times \gamma_I \ ,$$

где:

- β - коэффициент, характеризующий качество подзатворного окисла. Для толщины окисла 115 Å β =2,25 (чертеж 4 РД);

- N – количество элементов. N = 1200000;

- $\lambda_{эл}$ – интенсивность отказов одного элемента. Для МОП - транзисторов при VII степени интеграции $\lambda_{эл} = 3 \times 10^{-13}$ (1/ч) (таблица 8 РД);

- α_I - коэффициент режима, учитывающий влияние электрического режима и температуры. Для КМОП технологии он определяется в зависимости от коэффициента электрической нагрузки транзистора:

$$Kн = U_{уст}/(0{,}7\times U_{проб}) \ ,$$

где:

- $U_{уст}$ – установленное напряжение на затворе $U_{уст}$ = 10 В;

- $U_{проб}$ - пробивное напряжение затвора $U_{проб}$ = 16,5 В;

$Kн$ = 10/(0,7×16,5)=0,9

тогда из таблицы 9 РД α_I = 0,15;

- λ_M - интенсивность отказов металлизации единичной площади. Для металлических дорожек шириной (1÷3) мкм $\lambda_M = 0{,}16\times10^{-7}$ (1/ч) (таблица 8 РД);

- $S_м$ – площадь металлизации в мм^2, $S_м$ = 48 мм^2;

- γ_I - коэффициент, учитывающий влияние электрического режима и температуры на надежность металлизации, определяется в зависимости от отношения максимальной

фактической плотности тока к максимально допустимой (Q). Для данной ИС Q = 0,05, тогда

γ_I = 0,05 (чертеж 7 РД).

Подставив найденные значения в формулу для λ_2, получим:

$\lambda_2 = 2{,}25 \times 1{,}2 \times 10^{6} \times 3 \times 10^{-13} \times 0{,}15 + 0{,}16 \times 10^{-7} \times 48 \times 0{,}05 = 1{,}215 \times 10^{-7} + 0{,}384 \times 10^{-7} =$

$= 1{,}599 \times 10^{-7}$ (1/ч)

4 Следовательно, для нормальной температуры:

$\lambda_{ИС} = 0{,}2 \times (0{,}752 \times 10^{-7} + 1{,}599 \times 10^{-7}) = 0{,}47 \times 10^{-7}$ (1/ч)

В соответствии с пунктом 3.5.1 ТЗ, минимальная наработка до отказа микросхем рассчитывается при температуре 65 ºС. Значения $\lambda_{к}$, $\lambda_{кр}$, $\lambda_{с}$ при температуре 65 ºС увеличиваются в K_y раз, где K_y - коэффициент ускорения отказов, при повышении окружающей температуры с 25 ºС до 65 ºС, определяемый по номограмме приложения 8 чертеж 1 РД в зависимости от энергии активации отказов E_A (таблица 10 РД): для ИС в металлокерамических корпусах E_A=0,3 ЭВ, K_y=4; в случае посадки кристалла на клей: E_A=0,6 ЭВ, K_y=16; для соединений Al-Al: E_A=0,3 ЭВ, K_y=4. Тогда:

$\lambda_{к}$ (65 ºС)= $\lambda_{к} \times 4 = 0{,}28 \times 10^{-7} \times 4 = 1{,}12 \times 10^{-7}$ (1/ч); при E_A=0,3 ЭВ;

$\lambda_{кр}$ (65 ºС) = $\lambda_{кр} \times 16 = 0{,}01 \times 10^{-7} \times 16 = 0{,}16 \times 10^{-7}$ (1/ч) при E_A=0,6 ЭВ;

$\lambda_{с}$(65 ºС) = $\lambda_{с} \times 4 = 0{,}0015 \times 10^{-7} \times 4 = 0{,}006 \times 10^{-7}$ (1/ч) при E_A = 0,3 ЭВ.

Таким образом:

λ_1(65 ºС) = $1{,}62 \times 1{,}12 \times 10^{-7} + 3 \times 0{,}16 \times 10^{-7} + 0{,}8 \times 224 \times 0{,}006 \times 10^{-7} = 3{,}369 \times 10^{-7}$ (1/ч).

Значения α_I и γ_I для температуры 65 ºС определяем по таблице 9 и чертежу 7 РД соответственно: α_I (65 ºС) = 0,49; γ_I (65 ºС) = 0,4.

λ_2 (65 ºС) = $2{,}25 \times 1{,}2 \times 10^{6} \times 3 \times 10^{-13} \times 0{,}49 + 0{,}16 \times 10^{-7} \times 48 \times 0{,}4 =$

$= 3{,}969 \times 10^{-7} + 3{,}072 \times 10^{-7} = 7{,}041 \times 10^{-7}$ (1/ч);

Тогда:

$\lambda_{ИС}$ (65 ºС)=$0{,}2 \times (3{,}369 \times 10^{-7} + 7{,}041 \times 10^{-7}) = 2{,}082 \times 10^{-7}$ (1/ч);

5 Значение гамма - процентного ресурса (при γ = 0,99) определяем по формуле:

$T_{P\gamma} = (-ln(99/100)/\lambda_{ИС}$

$T_{P\gamma\,(65\,^{o}C)} = (-ln(99/100))/\lambda_{ИС}\,(65\,^{o}C) = 0{,}01005/\,(2{,}082\times10^{-7}) = 48271$ (ч).

6 Значение минимальной наработки определяем по формуле:

$T_{МН} = T_{P\gamma} \times ln(1-\alpha)\times(n\times ln(99/100))^{-1}$, где:

α - риск изготовителя, равный вероятности получения отрицательного результата при испытаниях на долговечность. Рекомендуемое РД значение $\alpha = 0{,}4$;

n - объем выборки для испытаний на долговечность. Согласно пункта 3.5.3 ТЗ на ОКР n=10;

$T_{НМ}(65\,^{o}C) = T_{P\gamma}(65\,^{o}C) \times ln(1-\alpha)\times(n\times ln(99/100))^{-1} = 48271\times0{,}51083\times(10\times0{,}01005)^{-1} =$ = 245355 (ч),

что удовлетворяет требуемому времени минимальной наработки 100000 часов в режимах и условиях, допускаемых ТЗ.

7 Определяем минимальный срок сохраняемости микросхем. Поскольку хранение проводят при нормальной окружающей температуре без подачи электрического режима, интенсивность отказов ИС при хранении:

$\lambda_{СХ} = K_{П} \times \lambda_{1} = 0{,}2\times0{,}752\times10^{-7}$(1/час) = $0{,}15\times10^{-7}$(1/ч)

Тогда минимальный срок сохраняемости микросхем:

$T_{СХ,\,\gamma} = -ln(99/100)\times(\lambda_{СХ})^{-1} = 0{,}01005/(0{,}15\times10^{-7}) = 670000$ (ч)

что удовлетворяет требуемому по ОСТ В 11 0998–99 минимальному сроку сохраняемости 25 лет или 219000 часов.

8 Для расчета зависимости интенсивности отказов ИС в зависимости от режимов и условий эксплуатации воспользуемся выражением:

$\lambda_{ИС}(Т) = K_{П} \times (\lambda_{1}(Т) + \lambda_{2}(Т))$

Поскольку, согласно пункта 2.2.5 РД, ускоряющим фактором для большинства механизмов отказов является повышенная температура, составляющие общей интенсивности отказов $\lambda_{1}(Т)$ и $\lambda_{2}(Т)$, зависящие от температуры, рассчитываются по формулам:

$\lambda_{1}(Т) = \alpha_{к}\times\lambda_{к}\times K_{у1}(Т) + \alpha_{кр}\times\lambda_{кр}\times K_{у2}(Т) + n\times\alpha_{с}\times\lambda_{с}\times K_{у1}(Т)$

$\lambda_{2}(Т) = \beta\times\lambda_{эл}\times N\times\alpha_{I}(Т) + \lambda_{М}\times S_{М}\times\gamma_{I}(Т)$

С учетом ранее выбранных значений коэффициентов $\alpha_{к}$, $\lambda_{к}$, $\alpha_{кр}$, $\lambda_{кр}$, n, $\alpha_{с}$, $\lambda_{с}$, β, $\lambda_{эл}$, N, $\lambda_{М}$, $S_{М}$ формулы для $\lambda_{1}(Т)$ и $\lambda_{2}(Т)$ принимают вид:

$\lambda_1(T) = (1{,}62\times0{,}28\times K_{y1}(T)+3\times0{,}01\times K_{y2}(T) + 0{,}8\times224\times0{,}0015\times K_{y1}(T)) \times10^{-7} =$

$= (0{,}722\times K_{y1}(T)+0{,}03\times K_{y2}(T))\times10^{-7}$ (1/ч)

$\lambda_2(T) = (2{,}25\times3\times1{,}2\times\alpha_I(T) + 0{,}16\times48\times\gamma_I(T)) \times10^{-7}$(1/ч) =

$= (8{,}1\times\alpha_I(T) + 7{,}68 \times\gamma_I(T)) \times 10^{-7}$(1/ч)

K_{y1} - коэффициент ускорения отказов для металлокерамического корпуса и соединений Al-Al (энергия активации - 0,3 ЭВ) при повышении окружающей температуры от плюс 25 ºС до плюс Т ºС (по номограмме чертёж 1 РД).

K_{y2} - коэффициент ускорения отказов для посадки кристалла на клей (энергия активации - 0,6 ЭВ) при том же повышении температуры (по номограмме чертеж 1 РД).

Значения α_I и γ_I при $K_н$ = 0,9 и Q=0,05 определяем по таблице 9 и чертежу 7 РД соответственно. Значения коэффициентов K_{y1}, K_{y2}, α_I, γ_I для различных значений окружающей температуры и соответствующие им значения интенсивности отказов $\lambda_{ИС}(T)$ сведены в таблицу 8.1.

Таблица 8.1

Т, ºС	K_{y1}	K_{y2}	α_I	γ_I	$\lambda_{ИС}(T)*10^{-7}$, 1/ч
1	2	3	4	5	6
25	1	1	0,15	0,05	0,470
35	1,5	2,0	0,21	0,095	0,715
45	2,0	4,0	0,29	0,16	1,028
55	3,0	8,5	0,38	0,24	1,468
65	4,0	16,0	0,49	0,40	2,082
75	5,3	27,0	0,68	0,58	2,920
85	6,9	46,0	0,95	0,93	4,240
95	9,0	80,0	1,27	1,40	5,987
105	12	110	1,61	2,1	8,227
115	16	200	1,94	2,9	11,108
125	20	350	2,15	4,0	14,615

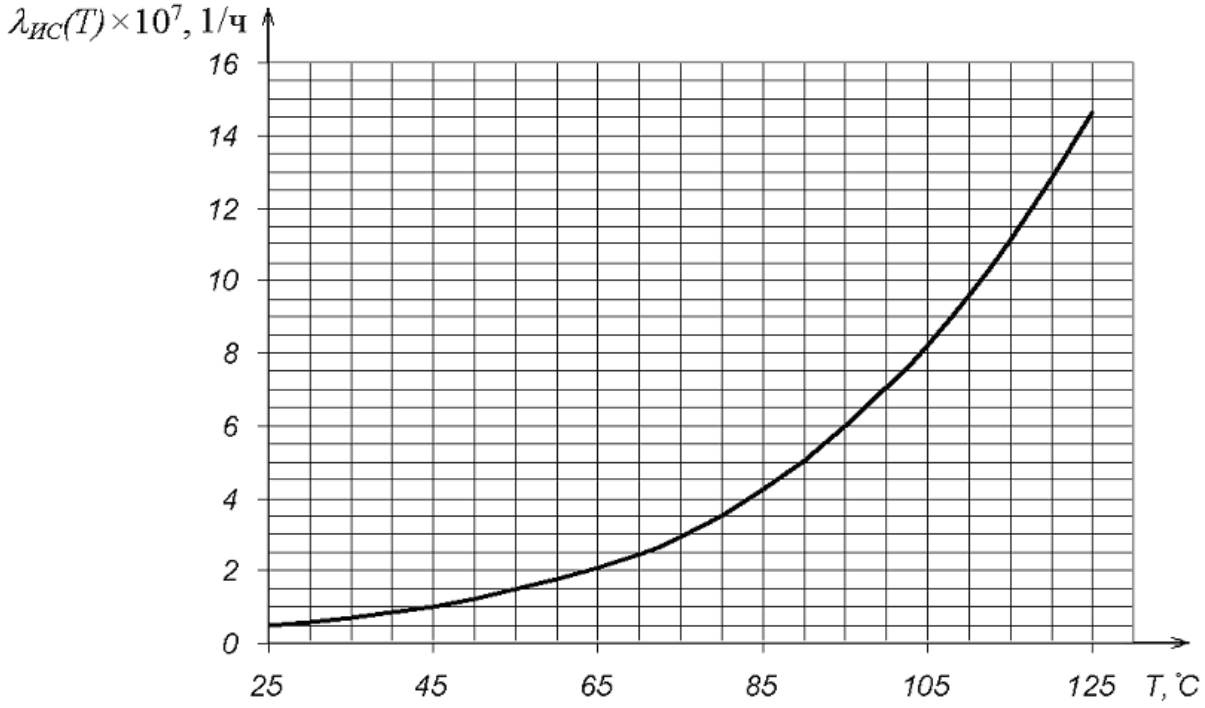


Рисунок 8.1 – Зависимость интенсивности отказов микросхем от изменения температуры кристалла

Помимо окружающей температуры, на температуру кристалла, а, следовательно, и на интенсивность отказов ИС, влияет электрический режим, что следует из формулы:

$T_{кр}=T_{окр}+R_T\times U_{CC}\times I_{CC}$,

где:

$T_{кр}$ - температура кристалла ИС, ºС,

$T_{окр}$ - температура окружающей среды, ºС,

R_T - тепловое сопротивление кристалл – окружающая среда, ºС/Вт. R_T=31,5 ºС/Вт.

U_{CC} - напряжение питания ИМС, В.

В предельно допустимом режиме U_{CC} =5,5 В.

I_{CC} - динамический ток потребления ИМС, А, определенный как сумма динамических токов потребления ядра и периферии при 65 ºС и тактовой частоте 10 МГц:

I_{CC} =(20,8+34) мА=54,8 мА.

Тогда:

$T_{кр}$=65 ºС + 31,5 ºС/Вт ×5,5 В ×54,8×10^{-3} А=74,5 ºС.

По графику (рисунок 8.1) зависимости $\lambda_{ИС}$(Т) можно определить значение

$\lambda_{ИС}$(74,5 ºС) = 2,9×10^{-7} (1/ч)

Тогда значение гамма - процентного ресурса (при γ = 0,99):

$T_{P\gamma\,(74,5\,^{o}C)} = (-ln(99/100))/\lambda_{ИС}\,(74,5\,^{o}C) = 0,01005/\,(2,9\times10^{-7}) = 34655$ (ч).

Значение минимальной наработки определяем по формуле:

$T_{НМ}(74,5\,^{o}C) = T_{P\gamma}\,(74,5\,^{o}C) \times ln(1-\alpha)\times(n\times ln(99/100))^{-1} =$

$$= 34655 \times \frac{0,51083}{10 \times 0,01005} = 176147 \text{ (ч)}$$

что удовлетворяет требуемому времени минимальной наработки 100000 часов в режимах и условиях, допускаемых техническим заданием.

В свою очередь I_{CC} зависит от напряжения питания U_{CC}, увеличение которого на 10 % вызывает увеличение I_{CC} на 20 %, а также от частоты тактового сигнала, увеличение которой в 2 раза увеличивает I_{CC} в 1,9 раза.

9 Заключение

В настоящем руководстве приведено описание архитектуры, функционального построения, системы команд и особенностей применения ИМС 1867ВЦ2Т (1867ВЦ2АТ), которая представляет собой СБИС 16-разрядного процессора цифровой обработки сигналов с фиксированной точкой и тактовой частотой 40 (57) МГц, со встроенным масочным ПЗУ 2К × 16 и встроенной оперативной памятью 10К × 16.

Все значения электрических параметров ИМС приведены в ТУ на изделие. Значения параметров, приведенные в руководстве, являются справочными.

Настоящее руководство может служить практическим пособием по применению процессоров ЦОС для разработчиков систем на основе ИМС 1867ВЦ2Т (1867ВЦ2АТ).

Применение процессора ЦОС в системах цифровой обработки сигналов, встроенных цифровых системах управления, связи, в системах автоматизации технологических процессов, вычислительной технике, телекоммуникационной технике и т.д. позволит создавать более совершенные в техническом отношении и надежные в эксплуатации изделия.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Подп.	Дата
	изме-ненных	заме-ненных	новых	анну-лиро-ванных				
1	-	9	-	-	403			21.08.07
2	-	6, 402	-	-	-			19.02.13