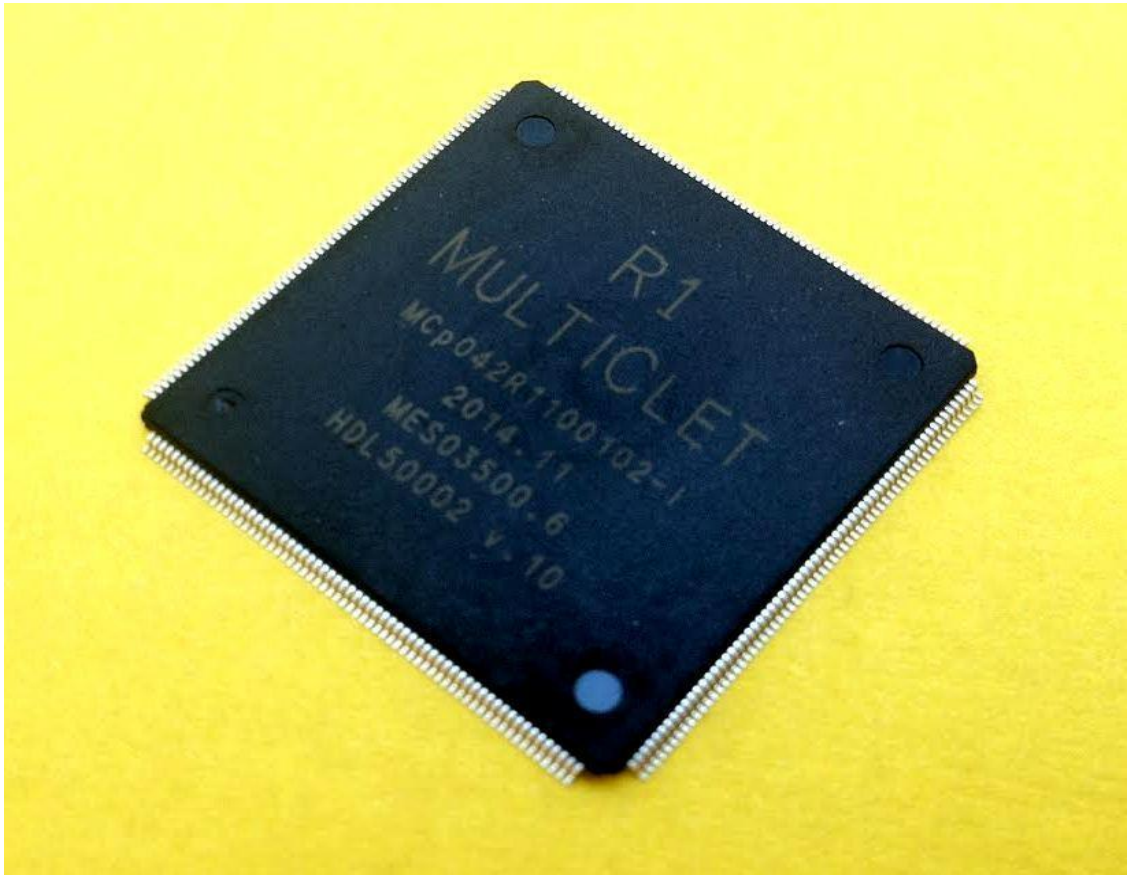
Программное обеспечение мультиклеточного процессора «MultiClet S1»

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ (V 1.0)



Оглавление

1. ОБЩИЕ СВЕДЕНИЯ 3
1.1. Запуск ассемблера и опции командной строки 3
1.2 Общие сведения о мультиклеточном процессоре 3
1.2.1 Типы данных и регистры управления 3
1.2.2 Форматы команд 6
1.3 Основные понятия языка 17
1.3.1 Комментарии 17
1.3.2 Константы 18
1.3.3 Секции 20
1.3.4 Символы 22
1.3.5 Выражения 24
1.4 Система команд ассемблера 26
1.4.1 Структура командного слова 26

1. ОБЩИЕ СВЕДЕНИЯ

1.1. Запуск ассемблера и опции командной строки

Ассемблер запускается из командной строки командой mc-as, аргументом которой является файл с исходным кодом.

1.2 Общие сведения о мультиклеточном процессоре

Процессор **MultiClet S1** предназначен для решения широкого круга научно-технических в приложениях, требующих минимального энергопотребления и высокой производительности.

Память для процессора **MultiClet S1** с программной точки зрения представляет собой виртуальный массив с байтовой адресацией и диапазоном адресов [0÷(32**2)-1]. Адреса разделены между памятью программ (данных) и периферийными устройствами.

Часть памяти программ (данных) отображается на внутреннюю память процессора, которая представляет собой статическое оперативное запоминающее устройство, размещенное на кристалле.

Внутренняя память разделена на два блока с независимым доступом, в одном из которых размещаются программы (PM), а во втором – данные (DM).

1.2.1 Типы данных и регистры управления

Типы данных Табл.1

Тип	Мнемоника	Размер (бит)	Код в поле F1	Тип в C
Байт	b	8	0001	signed char unsigned char
Короткое слово	s	16	0010	signed short unsigned short
Слово	l	32	0100	signed int unsigned int

Двойное слово	q	64	1000	signed long long unsigned long long
Число с плавающей запятой одинарной точности	f	32	0101	float
Число с плавающей запятой двойной точности	d	64	1001	double
4 коротких слова в упакованном формате	ps	64	1010	-
2 слова в упакованном формате	pl	64	1100	-
2 числа одинарной точности с плавающей запятой в упакованном формате	pf	64	1101	-

Регистры управления Табл.2

Наименование	Мнемоника	Номер регистра (NRg)	Назначение
Регистр состояния процессора	PSW	0	Регистр имеет следующую структуру: 0-й бит – признак состояния ядра, если данный бит «0», то ядро занято, иначе свободно*, признак формируется аппаратно, в исходном состоянии равен единице; 1-й бит – признак отмены очередности исполнения команд чтения/записи, если данный бит «0», то установлен режим контроля очередности исполнения команд чтения/записи, иначе контроль отменен, в исходном состоянии равен нулю; 2-й бит – признак готовности адреса следующего параграфа, если он равен «0», то адрес не готов, иначе готов**, в исходном состоянии признак равен нулю; 3-й бит –признак «complete». В исходном состоянии он равен нулю – предыдущий параграф не закончен. В «1» признак устанавливается при декодировании команды с признаком «complete» и отсутствии на данный момент адреса следующего параграфа (2-ой бит равен «0»),
Регистр адреса продолжения	RETA	1	Содержимое регистра формируется автоматически при обработке команды с признаком «complete». В регистр записывается адрес команды, следующей за командой с признаком. В исходном состоянии равен

			нулю.
Адрес следующего параграфа исполняе-мой программы	NXTP	2	Устанавливается программно. В исходном состоянии равен нулю.
Базовый адрес локаль-ных переменных ис-полняемой программы	BP	3	Устанавливается программно. В исходном состоянии равен нулю.
Указатель начального адреса свободной памяти данных	SP	4	Устанавливается программно. В исходном состоянии равен нулю.
Регистр возвращаемо-го значения	RETV	5	Устанавливается программно. В исходном состоянии равен нулю.
Регистр смещения адреса исполняемой программы	AOR	6	Устанавливается программно. В исходном состоянии равен нулю.
		7	
		8÷15	

Примечания:

* ядро считается свободным, если нет обращений в память программ, в буферах клеток нет команд и не сформирован адрес следующего параграфа.

** признак готовности адреса следующего параграфа формируется, если команды текущего параграфа еще не выбраны, а адрес уже сформирован. В этом случае адрес записывается в регистр NXTR а во второй бит PSW записывается единица.

1.2.2 Форматы команд

Командные слова процессора имеют следующие форматы:

CC - короткая команда 1 слово;

CA - короткая команда 1 слово, второй операнд которой константа;

CV – длинная команда 2 слова, второе слово которой содержит константу либо адрес аргумента;

VV – длинная команда 3 слова, второе и третье слова которой содержат константы либо адреса аргументов.

Выбор формата команды происходит по содержимому битов **F0.**

1.2.2.1 Сокращения и обозначения

ARG1 – значение первого аргумента выполняемой операции;

ARG2 – значение второго аргумента выполняемой операции;

TAG – тег команды (результата данной команды);

RAR – динамическая последовательность результатов выполненных команд (рассматривается как одномерный массив);

RAR(TAG) – результат команды с тегом TAG;

size - размер ARG2 в байтах, определяемый типом данных;

#<имя регистра|номер> - содержимое указанного регистра;

PM – память программ;

DM – память данных.

1.2.2.2 Форматы CC и CA

Бит	**31**	**…**	**25**	**24**	**23**	**22**	**…**	**19**	**18**	**17**	**…**	**12**	**11**	**…**	**6**	**5**	**…**	**0**
Назначение	**COP**			**F0**		**Type**			C	**-**			**F1**			**F2**		

COP - код операции;

F0 – определяет формат командного слова, а именно, **F0** =”00”, если команда имеет формат CC. **F0** = “01”, если команда имеет формат CA .

Type – тип данных;

C – признак завершения выборки команд текущего параграфа (complete);

F1 – разница между тегом данной команды и тегом команды результат которой используется операцией в качестве ARG1, либо номер регистра (в командах чтения/записи регистров):

ARG1 **:=** RAR(TAG – **F1**). Если содержимое **F1** равно нулю, то в качестве аргумента берется ноль**;**

F2 – разница между тегом данной команды и тегом команды результат которой используется операцией в качестве ARG2.

В командах формата CC:

ARG2 **:=** RAR(TAG – **F2**). Если содержимое **F2** равно нулю, то в качестве аргумента берется ноль**.** В командах сдвига на константу содержимое поля **F2** используется как константа, определяющая количество сдвигов.

В командах формата CA:

ARG2 := DM(RAR(TAG – **F2**))**.**

1.2.2.3 Формат CV

Бит	**31**	**…**	**25**	**24**	**23**	**22**	**…**	**19**	**18**	**17**	**…**	**12**	**11**	**…**	**6**	**5**	**…**	**0**
Назначение	**COP**			**F0**		**Type**			C	**-**			**F1**			**F2**		

Бит	**31**	**…**	**0**
Назначение	**V**		

ARG1 = RAR(TAG – **F1**). Если содержимое **F1** равно нулю, то в качестве аргумента берется ноль**;**

F0 – содержит “10”;

F2 определяет способ формирования второго аргумента, а именно:

Значение поля **F2**						Значение второго аргумента
5	4	3	2	1	0	
0	0	0	0	0	0	ARG2:= **V**
0	0	0	0	0	1	ARG2:=DM(**V**)
NRg					0	ARG2:=**#NRg** +**V**
NRg					1	ARG2:=DM(**#NRg +V**)

1.2.2.4 Формат VV

Бит	**31**	**…**	**25**	**24**	**23**	**22**	**…**	**19**	**18**	**17**	**…**	**12**	**11**	**…**	**6**	**5**	**…**	**0**
Назначение	**COP**			**F0**		**Type**			C	**-**			**F1**			**F2**		

Бит	**31**	**…**	**0**
Назначение	**V1**		

Бит	**31**	**…**	**0**
Назначение	**V2**		

F0 – содержит “11”;

F1 определяет способ формирования второго аргумента, а именно:

Значение поля **F1**						Значение первого аргумента
11	10	9	8	7	6	
0	0	0	0	0	0	ARG1:= **V**
0	0	0	0	0	1	ARG1:=DM(**V**)
NRg					0	ARG1:=**#NRg** +**V**
NRg					1	ARG1:=DM(**#NRg** +**V**)

F2 определяет способ формирования второго аргумента, а именно:

Значение поля **F2**						Значение второго аргумента
5	4	3	2	1	0	
0	0	0	0	0	0	ARG2:= **V**
0	0	0	0	0	1	ARG2:=DM(**V**)
NRg					0	ARG2:=**#NRg** +**V**
NRg					1	ARG2:=DM(**#NRg** +**V**)

1.2.2.4 Список команд

Инструкции доступа к памяти, регистрам и перехода Табл.3

№ п/п	**Выполняемая инструкция**	**Код**	**Мнемо-ника**	**Тип опе-ранда**	**Алгоритм**
Инструкция чтения					
	Загрузка знако-вого числа	1110000	LOAD	byte, short, long, quad,	RAR(TAG):=ARG2 Выполняется с контролем очередности доступа к памяти
	Загрузка беззна-кового числа	1110001	LOADU	byte, short, long, quad	RAR(TAG):=ARG2 Выполняется с контролем очередности доступа к памяти
	Прямая загрузка знакового числа	1111000	DLOAD	byte, short, long, quad	RAR(TAG):=ARG2 Выполняется без контроля очередности доступа к памяти (по готовности)
	Прямая загрузка беззнакового числа	1111001	DLOADU	byte, short, long, quad	RAR(TAG):= ARG2 Выполняется без контроля очередности доступа к памяти (по готовности)
	Чтение фраг-мента из внеш-	1110011	SRD	quad	

	ней памяти				
Инструкции записи					
	Чтение с одновременной записью	1110101	TAS	byte, ~~short, long, quad~~	RAR(TAG):=DM(ARG2) DM(ARG2) := ARG1 Выполняется с контролем очередности доступа к памяти
	Прямое чтение с одновременной записью	1111101	DTAS	byte, ~~short, long, quad~~	RAR(TAG):= DM(ARG2) DM(ARG2) := ARG1 Выполняется без контроля очередности доступа к памяти (по готовности)
	Запись в DM	1110110	WR	byte, short, long, quad	DM(ARG2) := ARG1 Выполняется с контролем очередности доступа к памяти
	Прямая запись в DM	1111110	DWR	byte, short, long, quad	DM(ARG2) := ARG1 Выполняется без контроля очередности доступа к памяти (по готовности)
	~~Модификация содержимого DM~~	~~1110111~~	~~MOD~~	~~byte, short, long, quad~~	~~DM(ARG2):= DM(ARG2)+ARG1 Выполняется с контролем очередности доступа к памяти~~
	~~Прямая модификация содержимого DM~~	~~1111111~~	~~DMOD~~	~~byte, short, long, quad~~	~~DM(ARG2):= DM(ARG2)+ARG1 Выполняется без контроля очередности доступа к памяти (по готовности)~~
	Запись фрагмента во внешнюю память	1110111	SWR	quad	

Инструкции для доступа к регистрам и перехода					
	Чтение регистра	1100100	GETRG	quad	RAR(TAG):= #F1
	Запись в регистр	1100000	SETRG	quad	#F1:= ARG2
	Модификация регистра	1100111	MODRG	long	#F1:= #F1 + ARG2
	Запись в регистр адреса следующего параграфа по	1100010	SETJT	long	Если значение первого аргумента «true» (т.е. не равно нулю), то формируемый адрес

	«true»				следующего параграфа равен значению второго аргумента, иначе адрес не формируется.
	Запись в регистр адреса следующего параграфа по «false»	1100011	SETJF	long	Если значение первого аргумента «false», (т.е. равно нулю) то формируемый адрес следующего параграфа равен значению второго аргумента, иначе адрес не формируется.
	Запись в регистр адреса следующего параграфа по «else»	1100001	SETJELSE	long	Если в текущем параграфе ни одна из команд записи в регистр адреса следующего параграфа не сформировала адрес следующего параграфа, то он устанавливается равным значению второго аргумента данной команды. В параграфе может быть только одна команда SETELSE.
	Запись в регистр адреса следующего параграфа по содержимому регистра (счетчика) с последующей модификацией содержимого	1101000	SETCNT	long	Если содержимое регистра (#F1) «true», (т.е. не равно нулю) то адрес следующего параграфа устанавливается равным значению второго аргумента, а содержимое регистра уменьшается на «1», иначе адрес не формируется, содержимое регистра не изменяется и устанавливается признак блокировки признака конца текущего параграфа, который обеспечивает выборку команд параграфа, непосредственно следующего за текущим параграфом.

Целочисленные арифметико-логические инструкции Табл.4

№ п/п	Выполняемая инструкция	Код	Мнемоника	Тип операнда	Алгоритм

Арифметические инструкции					
	Сложение	0000001	ADD	byte, short, long, quad, pack	RAR(TAG):=ARG1+ARG2
	Вычитание	0000010	SUB	byte, short, long, quad, pack	RAR(TAG):=ARG1-ARG2
	Формирование переноса	0011101	OF	quad	
	Формирование заёма	0011110	CR	quad	
	Выбор большего знакового числа	0000101	MAX	byte, short, long, quad pack	IF ARG1>ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Выбор большего беззнакового числа	0010101	MAXU	byte, short, long, quad pack	IF ARG1>ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Выбор меньшего знакового числа	0000110	MIN	byte, short, long, quad pack	IF ARG1<ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Выбор меньшего беззнакового числа	0010110	MINU	byte, short, long, quad pack	IF ARG1<ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Абсолютное значение	0000111	ABS	byte, short, long, quad pack	RAR(TAG):=|ARG2|
	Умножение знаковых чисел	0000011	MUL	long, pack	RAR(TAG):=ARG1*ARG2
	Умножение беззнаковых чисел	0010011	MULU	long, pack	RAR(TAG):=ARG1*ARG2
	Деление знакового числа	0000100	DIVREM	long	RAR(TAG):=ARG1/ARG2 Результат деления в битах 31:0, а остаток от деления в

					битах 63:32.
	Деление беззнаковоого числа	0010100	DIVREMU	long	RAR(TAG):=ARG1/ARG2 Результат деления в битах 31:0, а остаток от деления в битах 63:32.
	Сравнение знаковых чисел по GE	0001000	GE	byte, short, long, quad pack	Если первый аргумент больше или равен второму, то результат «true», иначе «false»
	Сравнение знаковых чисел по LT	0001001	LT	byte, short, long, quad pack	Если первый аргумент меньше второго, то результат «true», иначе «false»
	Сравнение беззнаковых чисел по GE	0011000	GEU	byte, short, long, quad pack	Если первый аргумент больше или равен второму, то результат «true», иначе «false»
	Сравнение беззнаковых чисел по LT	0011001	LTU	byte, short, long, quad pack	Если первый аргумент меньше второго, то результат «true», иначе «false»
Логические инструкции					
	Склейка числа	0100000	PATCH	quad	RAR(TAG):=ARG1(31÷0)& ARG2(31÷0)
	Упаковка числа	0100001	PACK	quad	RAR(TAG):= ARG1(31÷0)&ARG2(63÷32)
	«И»	0100010	AND	byte, short, long, quad	RAR(TAG):=ARG1&ARG2
	«ИЛИ»	0100011	OR	byte, short, long, quad	RAR(TAG):=ARG1|ARG2
	«Исключающее ИЛИ»	0100100	XOR	byte, short, long, quad	RAR(TAG):=ARG1^ARG2
	«НЕТ»	0100101	NOT	byte, short, long, quad	RAR(TAG):= ~ARG2

	«И-НЕТ»	0110010	ANDNOT	byte, short, long, quad	RAR(TAG):=ARG1& (~ARG2)
	«Исключающее ИЛИ» со сдви-нутым вторым аргументом	0110100	XORROL1	byte, short, long, quad	RAR(TAG):=ARG1^ROL (ARG2,1)
	Сканирование битов вперёд	0100110	BSF	long, quad	Вычисляется порядковый номер первого бита, содержимое которого равно единице, обнаруженного при сканировании ARG2 в направлении от младшего, 0-го бита к старшему 31(63)-му биту. Порядковый номер определяется, как номер бита, увеличенный на единицу. Если ARG2 равно нулю, то результат равен нулю.
	Сканирование битов назад	0100111	BSR	long, quad	Вычисляется порядковый номер первого бита, содержимое которого равно единице, обнаруженного при сканировании ARG2 в направлении от старшего бита 31(63)-го к младшему 0-му. Порядковый номер определяется, как номер бита, увеличенный на единицу. Если ARG2 равно нулю, то результат равен нулю.
	Логический сдвиг числа влево	1010000	SLL	~~byte,~~ short, long, quad, pack	RAR(TAG):=ARG1<<ARG2 Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Логический сдвиг числа вправо	1010001	SLR	~~byte,~~ short, long, quad, pack	RAR(TAG):=ARG1>>ARG2 Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Арифметический сдвиг числа	1010010	SAR	~~byte,~~ short,	RAR(TAG):=ARG1/ (2**ARG2)

	вправо			long, quad, pack	Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Циклический сдвиг числа влево	1010011	ROL	byte, short, long, quad, pack	RAR(TAG):=ROL(ARG1, ARG2) Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Циклический сдвиг числа вправо	1010100	ROR	~~byte,~~ short, long, quad, pack	RAR(TAG):= ROR(ARG1, ARG2) Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Логический сдвиг числа влево на константу	1011000	SLLCN	~~byte,~~ short, long, quad	RAR(TAG):=ARG1<<F2 Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Логический сдвиг числа вправо на константу	1011001	SLRCN	~~byte,~~ short, long, quad	RAR(TAG):=ARG1>>F2 Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Арифметический сдвиг числа вправо на константу	1011010	SARCN	~~byte,~~ short, long, quad	RAR(TAG):=ARG1/(2**F2) Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Циклический сдвиг числа влево на константу	1011011	ROLCN	~~byte,~~ short, long, quad	RAR(TAG):=ROL(ARG1, F2) Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Циклический сдвиг числа вправо на константу	1011100	RORCN	~~byte,~~ short, long, quad	RAR(TAG):=ROR(ARG1, F2) Количество сдвигов (значение ARG2) должно быть меньше размерности сдвигаемого числа.
	Выбор второго аргумента по «true»	0101000	ST	byte, short, long, quad	Если значение первого аргумента «true», то результат операции равен значению второго

					аргумента, иначе нулю.
	Выбор второго аргумента по «false»	0101001	SF	byte, short, long, quad	Если значение первого аргумента «false», то результат операции равен значению второго аргумента, иначе нулю.
	Вычисление числа битов, равных единице	0101010	BC	long, quad	Суммируется количество единиц в ARG2
	Развёртка байтов с размножением знака	0101011	EXPBS	long	
	Развёртка байтов	0101100	EXPB	long	
	Свёртка байтов с насыщением	0101101	PCKBS	quad	
	Свёртка байтов	0101110	PCKB	quad	
	Перестановка байтов	0101111	BSWAP	byte, short, long, quad, pack	

Арифметические инструкции с плавающей точкой Табл.5

№ п/п	**Выполняемая инструкция**	**Код**	**Мнемо-ника**	**Тип опе-ранда**	**Алгоритм**
Арифметические инструкции					
	Сложение	0000001	ADD	float, double, pack	RAR(TAG):=ARG1+|ARG2
	Вычитание	0000010	SUB	float, double, pack	RAR(TAG):=ARG1-ARG2
	Умножение	0000011	MUL	float, double, pack	RAR(TAG):=ARG1*|ARG2
	Комплексное умножение	0001011	CMUL	pack	RAR(TAG):=ARG1*|ARG2 ARG1 и ARG2 являются числами типа pf, каждое из которых состоит

					действительной и мнимой части типа f
	Деление	0000100	DIV	float, double	RAR(TAG):=ARG1/ARG2
	Извлечение квадратного корня	0001010	SQRT	float, double	RAR(TAG):=SQRT(ARG2)
	Выбор большего числа	0000101	MAX	float, double pack	IF ARG1>ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Выбор меньшего числа	0000110	MIN	float, double pack	IF ARG1<ARG2 THEN RAR(TAG):=ARG1 ELSE RAR(TAG):=ARG2
	Абсолютное значение	0000111	ABS	float, double pack	RAR(TAG):=|ARG|
	Сравнение чис-ла по GE	0001000	GE	float, double pack	Если первый аргумент больше или равен второму, то результат «true», иначе «false»
	Сравнение чис-ла по LT	0001001	LT	float, double pack	Если первый аргумент меньше второго, то результат «true», иначе «false»
Инструкции преобразования типов					
	Преобразование signed int в float	0001100	CIF	long	
	Преобразование float в signed int	0001101	CFI	float	
	Преобразование float в double	0001110	CFD	float	
	Преобразование double в float	0001111	CDF	double	

1.3 Основные понятия языка

1.3.1 Комментарии

В ассемблере поддерживаются следующие типы комментариев:

• однострочный комментарий, который начинается с ’;’ или ’//’ и заканчивается концом строки;

• многострочный комментарий, который начинается с ’/*’ и заканчивается ’*/’, т. е. все, что находится между этими ограничителями игнорируется.

Вложенные комментарии не допускаются.

1.3.2 Константы

В ассемблере поддерживаются числовые и символьные (литеральные) константы.

1.3.2.1 Числовые константы

В ассемблере возможны следующие варианты представления числовых констант:

1. В виде шестнадцатеричного числа, которое начинается с префикса ’0x’ или ’0X’, за которым следует одна или более шестнадцатеричных цифр ’0123456789abcdefABCDEF’. Для изменения знака используется префиксный оператор ’-’.

2. В виде восьмеричного числа, начинающегося с нулевой цифры, за которой следует одна или более восьмеричных цифр ’01234567’. Для изменения знака используется префиксный оператор ’-’.

3. В виде двоичного числа. Такое число начинается с префикса ’0b’ или ’0B’, за которым следует одна или более двоичных цифр ’01’. Для изменения знака используется префиксный оператор ’-’.

4. В виде целого десятичного числа, которое начинается с ненулевой цифры, за которой следует одна или более десятичных цифр ’0123456789’. Для изменения знака используется префиксный оператор ’-’.

5. В виде вещественного десятичного числа с плавающей точкой, записанного в следующем формате:

а) начинается с префикса ’0f’ или ’0F’,

б) далее опционально следует знак числа ’+’ или ’-’,

в) далее опционально следует целая часть числа, состоящая из нуля или более десятичных цифр,

г) далее опционально следует дробная часть числа, начинающаяся с символа точки ’.’ и состоящая из нуля или более десятичных цифр,

д) далее опционально следует экспоненциальная часть числа, состоящая из:

- ’e’ или ’E’
- знака ’+’ или ’-’ экспоненциальной части (опционально)
- одной или более десятичных цифр.

По крайней мере одна из целой или дробной частей должна быть задана.

1.3.2.2 Символьные (литеральные) константы

В ассемблере возможны следующие варианты представления символьных (литеральных) констант:

1. В виде строки (последовательности литералов), записанной в двойных кавычках. Они могут содержать любые возможные символы (литералы), а также следующие escape-последовательности:

- \b забой (backspace); ASCII код в восьмеричной системе счисления 010
- \f новая страница (FormFeed); ASCII код в восьмеричной системе счисления 014
- \n перевод строки (newline); ASCII код в восьмеричной системе счисления 012
- \r возврат каретки (carriage-Return); ASCII код в восьмеричной системе счисления 015
- \t горизонтальная табуляция (horizontal Tab); ASCII код в восьмеричной системе счисления 011
- \ oct-digit oct-digit oct-digit код символа в восьмеричной системе счисле-ния. Код символа состоит из 3-х восьмеричных цифр. Если заданное число превышает максимально возможное восьми разрядное значение, будут использованы только младшие восемь разрядов
- \x hex-digit hex-digit код символа в шестнадцатеричной системе счисления. Код символа состоит из 2-х шестнадцатеричных цифр. Регистр литерала ’x’ не имеет значения. Если ни одна из двух шестнадцатеричных цифр на задана, используется значение ноль

- \\ соответствует литералу ’\’
- \, соответствует литералу ’,’
- \anything-else соответствует любому символу, за исключением выше перечисленных.

2. В виде одиночного символа (литерала)

Одиночный символ (литерал) может быть представлен в виде одинарной кавыч-ки «´», непосредственно за которой следует необходимый символ (литерал) или escape-последовательность. Поэтому для представления символа (литерала) «\», необходимо написать «\\», где первый символ (литерал) «\» экранирует второй «\». Символ перевода строки, непосредственно следующий за «´», интерпретируется как символ (литерал) «\n» и не является окончанием выражения. Значением символьной (литеральной) константы в целочисленном выражении является машинный код, размером в один байт, символа (литерала). as использует ASCII кодировку символов (литералов): ´A имеет целочисленное значение 65, ´B имеет целочисленное значение 66, и так далее.

1.3.3 Секции

Секция представляет собой непрерывный диапазон адресов, все данные которого трактуются одинаково для некоторых определенных действий.

В результате компиляции исходного кода части программы, ассемблер создает объектный файл, предполагая, что данная часть программы располагается с нулевого адреса. Сборка самой исполняемой программы осуществляется компоновщиком из одного или нескольких объектных файлов, созданных ассемблером, в результате чего каждому объектному файлу присваивается конечный адрес таким образом, что ни один объектный файл не перекрывается другим.

Во время компоновки программы блоки байтов, как единое целое, перемещаются на те адреса, которые они будут иметь во время выполнения программы; их длина и порядок байтов в них не изменяются. Именно такой блок байтов называется секцией, а процедура назначения адресов этим секциям перемещением

(relocation). Помимо назначения секциям адресов времени выполнения и их перемещения, на этапе компоновки приводятся в соответствие значения символов объектного файла, так чтобы все ссылки на эти символы имели верные адреса времени выполнения.

Объектный файл, созданный ассемблером, включает в себя, по крайней мере, три секции, каждая из которых может быть пустой:

1. секция .text – в ней располагаются исполняемые инструкции программы.
2. секция .data, в которой располагаются начальные данные программы.
3. секция .bss. Данная секция содержит байты с нулевыми значениями перед началом выполнения программы. Она используется для хранения неинициализированных переменных или общего блока памяти данных.

Секции .text и .data присутствуют в объектном файле в не зависимости от того содержат они какие-либо директивы или нет.

Для того, чтобы сообщить компоновщику какие данные изменяются во время перераспределения памяти (перемещения секций), а также согласно каким правилам они изменяются, ассемблер записывает в объектный файл всю необходимую информацию в отдельные секции (как правило для секции .text в секцию .rel.text, для секции .data в секцию .rel.data).

Кроме секций .text, .data, .bss возможно использование абсолютной секции. При компоновке программы адреса в абсолютной секции не изменяются.

Помимо выше перечисленных секций существует также неопределённая секция. Любой символ, на который имеется ссылка и, который не был определен, на этапе ассемблирования относится к неопределенной секции. Общий (совместно используемый) символ, который адресует именованный общий блок, также на этапе ассемблирования относится к неопределенной секции. Значение атрибута связывания любого неопределённого символа по умолчанию равно «GLOBAL».

1.3.3.1 Подсекции

Ассемблированные байты по умолчанию размещаются в двух секциях: .text и .data. Для упорядочения различных групп данных внутри секций .text или .data

в генерируемом объектном файле используются подсекции. Внутри каждой секции могут находиться пронумерованные подсекции, начиная с нуля. Объекты, ассемблированные в одну и ту же подсекцию, в объектном файле располагаются вместе с другими объектами той же подсекции.

Подсекции располагаются в объектном файле в порядке возрастания их номеров. Информация о подсекциях в объектном файле не сохраняется.

Для того, чтобы указать в какую подсекцию ассемблировать ниже следующие инструкции, необходимо указать номер подсекции в директиве .text/.data. Если в исходном коде программы подсекции не используются, то все инструкции ассемблируются в подсекцию с номером 0.

Каждая секция имеет «счетчик текущего места», увеличивающийся на один при ассемблировании каждого нового байта в эту секцию. Поскольку подсекции являются просто удобством и ограничены использованием только внутри ассемблера, не существует «счетчиков текущего места» подсекций. «Счетчик текущего места» секции, в которую в данный момент ассемблируются инструкции, называется активным счетчиком места.

Секция bss используется как место для хранения глобальных переменных. Для этой секции, используя директиву .lcomm, можно выделить адресное пространство без указания какие данные будут загружены в нее до исполнения программы. Во время начала выполнения программы содержимое секции .bss заполняется нулями.

1.3.4 Символы

Символы (идентификаторы, переменные и т. п.) используются в ассемблере для именования различных сущностей.

1.3.4.1 Система имён символов

Система имён в ассемблере построена по следующему принципу имена могут состоять только из прописных и печатных букв латинского алфавита, цифр, символа подчеркивания «_» и символа точки «.», при этом имя не может начинаться с цифры. Учитывается регистр букв в имени.

1.3.4.2 Метки

Метка определяется как символ, за которым следует двоеточие ¾:¿. Этот символ представляет текущее значение активного счетчика места в зависимости от текущей секции (.text, .data, .bss). Переопределение меток в ассемблере не допускается.

1.3.4.3 Символы с абсолютным значением

Данные символы могут быть определены при помощи следующих директив ассемблера: .set. .eqv, .equ, .equiv. Значения данных символов никогда не изменяются компоновщиком. В общем случае, символ, определённый при помощи перечисленных директив ассемблера, может не иметь абсолютного значения.

1.3.4.4 Атрибуты символов

Каждый символ помимо имени имеет атрибуты «Значение», «Тип/Связывание», «Размер», а также атрибут принадлежности символа к какой-либо секции. При использовании символа без его определения все его атрибуты имеют нулевые значения, а сам символ относится к неопределенной секции.

Значение символа является 32-х разрядным. Для символов, адресующих местоположение в секциях .text, .data, .bss, а также абсолютной, значением является смещение в адресах относительно начального положения секции до метки. В процессе компоновки программы значения таких символов для секций .text, .data, .bss изменяются, поскольку изменяются начальные положения данных секций. Значения абсолютных символов в процессе компоновки не изменяются.

Атрибут символа «Тип/Связывание» определяет тип и видимость символа компоновщиком, а также поведение компоновщика в процессе изменения значения символа при перемещении секций. Для установки значения типа данного атрибута используется директива ассемблера .type, а для значения связывания директивы ассемблера .local, .global, .weak.

Атрибут символа ¾Размер¿ на этапе ассемблирования по умолчанию всегда устанавливается равным нулю. Значение данного атрибута может быть изменено при помощи директивы ассемблера .size.

Атрибут принадлежности символа к какой-либо секции устанавливается ассемблером автоматически, в зависимости от текущей секции ассемблирования. Другими словами данный атрибут указывает, в какой секции определён символ.

1.3.5 Выражения

Выражение определяет адрес или числовое значение. Результат выражения должен быть абсолютным числом или смещением в определенной секции.

1.3.5.1 Пустые выражения

Пустое выражение не имеет значения: это просто пропуск. В случае отсутствия выражения в том месте исходного кода, где оно необходимо, ассемблер использует значение ноль.

1.3.5.2 Целочисленные выражения

Целочисленное выражение это один или более аргументов, разделенных операторами.

1.3.5.3 Аргументы

В качестве аргументов выражения могут выступать символы (идентификаторы), числа или подвыражения.

Значением символа в какой-либо секции secName является смещение относительно начала этой секции. В качестве секции secName могут выступать секции .text, .data, .bss, а также абсолютная (*ABS*) и неопределённая (*UND*) секции. Значение представляет собой 32-х разрядное целое число со знаком в двоичном дополнительном коде.

Числа, как правило, являются целыми. Если выражение состоит из одного аргумента, который представляет собой число с плавающей точкой, то результатом вычисления выражения будет именно это число без каких-либо преобразований. Если выражение состоит из нескольких аргументов, разделенных операторами, либо из префиксного оператора, за которым следует один единственный аргумент, то при вычислении выражения значение аргумента, которое представ-

ляет собой число с плавающей точкой, будет заменено на нулевое значение, а также выведено соответствующее предупреждение.

Подвыражения представляют из себя такие же выражения, заключенные в круглые скобки «( )», либо префиксный оператор, за которым следует аргумент.

1.3.5.4 Операторы

Операторы представляют из себя арифметические функции и делятся на префиксные и инфиксные.

Префиксные операторы являются одноаргументными. Аргумент должен быть абсолютным. Доступны следующие префиксные операторы:

«–» Отрицание. Двоичное дополнительное отрицание.

«~» Дополнение. Побитовое отрицание.

«!» Логическое НЕ. Возвращается 1, если аргумент не нулевой, и 0, в противном случае.

Инфиксные операторы являются двухаргументными. Данные операторы имеют приоритет, который определяет очередность их выполнения. Операции с равным приоритетом выполняются слева на право. Аргументы всех инфиксных операторов, за исключением операторов «+» и «–» должны быть абсолютными. Доступны следующие инфиксные операторы в порядке снижения приоритета:

1. «*» Умножение.
   «/» Целочисленное деление.
   «%» Остаток (взятие остатка от целочисленного деления).
   «<<» Сдвиг влево.
   «>>» Сдвиг вправо.
2. «|» Побитовое Или.
   «&» Побитовое И.
   «^» Побитовое Исключающее Или.
   «!» Побитовое Или-Не
3. «+» Сложение. Если один из аргументов абсолютный, то результат относится к секции другого аргумента. Сложение двух аргументов, определенных относительно различных секций недопустимо.

«–» Вычитание. Если правый (второй) аргумент абсолютен, то результат относится к секции левого (первого) аргумента. Если оба аргументы определены относительно одной и той же секции, то результат абсолютен. Вычитание двух аргументов, определенных относительно различных секций недопустимо.

«==» Сравнение на равенство.

« =» или «< >» Сравнение на неравенство.

«<» Сравнение на меньше, чем.

«>» Сравнение на больше, чем.

«>=» Сравнение на больше, чем, либо равно.

«<=» Сравнение на меньше, чем, либо равно.

В качестве результата выполнения какой-либо операции сравнения, возвращается -1, в случае если результат истинный, и 0, если ложный. Операции сравнения интерпретируют аргументы как знаковые.

4. «&&» Логическое И.

«||» Логическое ИЛИ.

Данные логические операции могут быть использованы для объединения результатов двух подвыражений. В результате выполнения какой-либо логической операции, возвращается 1, в случае если результат истинный, и 0, если ложный.

1.4 Система команд ассемблера

В процессоре **MultiClet S1** существуют короткие команды, размерностью 32 бита (форматы CC и CA), и длинные, размерностью 64 (формат CV) или 96 бит (формат VV).

1.4.1 Структура командного слова

Командное слово ассемблера имеет следующую структуру:

<метка>**:**<мнемокоп>_<тип данных> <ARG1>**,**<ARG2>**;**

<метка>**:** метка не является обязательным элементом командного слова ассемблера и используется, если данное командное слово является первым словом параграфа, либо если использование его результата в других командах задается меткой;

<мнемокоп>_ наименование выполняемой операции в соответствии с системой команд (см. табл.2), заканчивающее подчерком;

<тип данных> размерность и тип используемых данных(см. табл.1), а именно:

b – знаковые или беззнаковые байты;

s – знаковые или беззнаковые 16-ти битные слова;

l – знаковые или беззнаковые 32-х битные слова;

q – знаковые или беззнаковые 64-х битные слова;

f – 32-х битные слова типа float;

df –64-х битные слова типа double float;

ps – упакованные 16-ти битные слова;

pl – упакованные 32-х битные слова;

pf – упакованные 32-х битные слова типа float.

В командах форматов CC CA и CV ARG1 имеет следующий вид:

@<метка>

где <метка> - метка команды, результат которой используется в качестве первого операнда, или:

@<число>

где <число> - целое число в диапазоне 1-63 равное количеству команд между командой источником результата и командой потребителем его в качестве первого аргумента выполняемой операции.

ARG2 в командах формата CC присутствует всегда и записывается аналогично операнду 1.

ARG2 в командах формата CA присутствует всегда и записывается следующим образом [@<метка>|@<число>].

В командах форматов CV и VV, ARG2 (формат CV) или ARG1 и ARG2 (формат VV) могут быть:

- константой, значение которой может рассматриваться как собственно константа, либо как смещение относительно регистра BP или SP. В последнем случае она записывается как #BP + const или #SP + const;
- символом;

– содержимым памяти, адрес которого задается символом и записывается в квадратных скобках, например:

[A+4] ,

где, A — метка в секции данных, например, .data.

В отдельных командах значение ARG1 и ARG2 может задаваться непосредственно малоразмерной константой, размещенной в полях F1 и F2. Например, в командах чтения и записи регистров, в командах сдвига на константу.