# ST10 FAMILY PROGRAMMING MANUAL

2nd EDITION AUGUST 1997



# Table of Contents -

\_\_\_\_\_

1	INT	RODUCTION
	ST	ANDARD INSTRUCTION SET
2	AD	DRESSING MODES
	2.1	Short addressing modes7
	2.2	Long addressing mode9
	2.3	Indirect addressing modes
	2.4	Constants
	2.5	Branch target addressing modes
3	INS	TRUCTION EXECUTION TIMES14
	3.1	Definition of measurement units14
	3.2	Minimum state times
	3.3	Additional state times
4	INS	TRUCTION SET SUMMARY
	4.1	X-ref tables of opcode by mnemonic
	4.2	X-ref table of mnemonic, address mode & number of bytes
	4.3	Instruction set ordered by functional group
	4.4	Instruction set ordered by opcodes
	4.5	Instruction conventions
	4.6	Notes on ATOMIC and EXTended instructions
5	IND	VIVIDUAL INSTRUCTION DESCRIPTIONS
	ADD	
	ADD	DB
	ADD	DC
	ADD	DBC
	ANE	
	AND	DB
	AS⊦	IR
	ATC	MIC
	BAN	ID
	BCL	.R
	BCN	۱P
	BFL	DH
	BFL	DL
	BMC	DV
	BMC	OVN

# Table of Contents

\_\_\_\_\_

BOR
BSET
BXOR
CALLA
CALLI
CALLR
CALLS
CMP
СМРВ
CMPD1
CMPD2
CMPI1
CMPI2
CPL
CPLB
DISWDT
DIV
DIVL
DIVLU
DIVU
EINIT
EXTR
EXTP
EXTPR
EXTS
EXTSR
IDLE
JB
JBC
JMPA
JMPI
JMPR
JMPS
JNB
JNBS

# Table of Contents -

\_\_\_\_\_

MOV	99
MOVB	.100
MOVBS	.101
MOVBZ	.102
MUL	.103
MULU	.104
NEG	.105
NEGB	.106
NOP	.107
OR	.108
ORB	.109
PCALL	.110
POP	.111
PRIOR	.112
PUSH	.113
PWRDN	.114
RET	.115
RETI	.116
RETP	.117
RETS	.118
ROL	.119
ROR	.120
SCXT	.121
SHL	.122
SHR	.123
SRST	.124
SRVWDT	.125
SUB	.126
SUBB	.127
SUBC	.128
SUBCB	.129
TRAP	.130
XOR	.131
XORB	.132

# ——— Table of Contents ——

	MAC INSTRUCTION SET
6	MAC ADDRESSING MODES
7	MAC INSTRUCTION EXECUTION TIME
8	MAC INSTRUCTION SET SUMMARY135
	8.1 MAC instruction conventions
9	INDIVIDUAL INSTRUCTION DESCRIPTION
	CoMUL(-)
	CoMULu(-)
	CoMULsu(-)
	CoMULus(-)
	CoMAC(R/-)
	CoMAC(R)u(-)
	CoMAC(R)su(-)
	CoMAC(R)us(-)
	CoMACM(R/-)
	CoMACM(R)u(-)
	CoMACM(R)su(-)
	CoMACM(R)us(-)
	CoADD(2)
	CoSUB(2)(R)
	CoNEG
	CoABS
	CoLOAD(2)(-)
	CoNOP
	CoSHL
	CoSHR
	CoASHR
	CoRND
	CoMAX
	CoMIN
	CoCMP
	CoSTORE
	CoMOV

# 1 Introduction

This programming manual details the instruction set for the ST10 Family of products. The manual is arranged in two sections. Section 1 details the standard instruction set and includes all of the basic instructions. Section 2 details the extension to the instruction set provided by the MAC co-processor. The MAC instructions are only available to devices containing the MAC co-processor, refer to the datasheet for device specific information.

Section 1 of this manual describes the different addressing modes used by the instruction set to access word, byte and bit data. Instruction execution times, minimum state times and the causes of additional state times are defined. Cross reference tables of instruction mnemonics, hexadecimal opcode, address modes and number of bytes, can be used as a quick reference for the optimization of instruction sequences. Tables of the instruction set ordered by functional group, defining the mnemonic, a brief instruction description and the minimum state time for all configurations have been included. These can be used for quick identification of the best instruction for a required function. A table of the instruction set ordered by hexadecimal opcode can be used to identify specific instructions when reading executable code i.e. during the de-bugging phase. Finally, each instruction is described individually on a page of standard format and using the conventions defined in this manual. For ease of use, the instructions are listed alphabetically.

The ATOMIC and EXTended instructions are not available to the ST10X166. This device is one of the older members of the ST10 family and does not have the Extended Special Function Register Area. This has been noted throughout the manual where applicable.

The MAC instruction set is divided into 5 functional groups: Multiply and Multiply-Accumulate Instructions, 32-Bit Arithmetic Instructions, Shift Instructions, Compare Instructions and Transfer Instructions. Two new addressing modes have been added to the MAC instruction set, supplying the MAC with up to 2 new operands per instruction. These addressing mode have been described in the MAC section. Cross reference tables of MAC instruction mnemonics by address mode, and MAC instruction mnemonic by functional code have been included for quick reference. As for the standard instruction set, each instruction has been described individually in a standard format, according to the defined conventions. In the MAC section the instructions are not described in alphabetical order, but by functional group.

<u>ل</u>حک

# **SECTION 1: STANDARD INSTRUCTION SET**

# 2 Addressing Modes

The ST10 family of devices use several powerful addressing modes for access to word, byte and bit data. This section describes short, long and indirect address modes, constants and branch target addressing modes.

## 2.1 Short addressing modes

Short addressing modes use an implicit base offset address to specify the physical address. For the ST10X166, the physical address is 18-bit, and for all other devices it is 24-bit.

Short addressing modes give access to the GPR, SFR or bit-addressable memory space:

#### Physical Address = Base Address + $\Delta$ \* Short Address

Note:  $\Delta$  is 1 for byte GPRs,  $\Delta$  is 2 for word GPRs.

Mnemonic	Physi	cal Address	Short	Address Range		Scope of Access
Rw	(CP)	+ 2*Rw	Rw	= 015	GPRs	(Word) 16 values
Rb	(CP)	+ 1*Rb	Rb	= 015	GPRs	(Byte) 16 values
reg	00'FE00h	+ 2*reg	reg	= 00hEFh	SFRs	(Word, Low byte)
	00'F000h	+ 2*reg <sup>*)</sup>	reg	= 00hEFh	ESFR	s(Word, Low byte) <sup>*)</sup>
	(CP)	+ 2*(reg∧0Fh)	reg	= F0hFFh	GPRs	(Word) 16 values
	(CP)	+ 1*(reg∧0Fh)	reg	= F0hFFh	GPRs	(Bytes) 16 values
bitoff	00'FD00h	+ 2*bitoff	bitoff	= 00h7Fh	RAM	Bit word offset 128 values
	00'FF00h	+ 2*(bitoff \FFh)	bitoff	= 80hEFh	SFR	Bit word offset 128 values
	(CP)	+ 2*(bitoff∧0Fh)	bitoff	= F0hFFh	GPR	Bit word offset 16 values
bitaddr	Word offs	et as with bitoff.	bitoff	= 00hFFh	Any sir	ngle bit
	Immediate	e bit position.	bitpos	= 015		

 Table 2.1
 Short addressing mode summary

Note: \* The Extended Special Function Register (ESFR) area is not available in ST10X166 devices.

#### Addressing Modes

- Rw, Rb: Specifies direct access to any GPR in the currently active context (register bank). Both 'Rw' and 'Rb' require four bits in the instruction format. The base address of the current register bank is determined by the content of register CP. 'Rw' specifies a 4-bit word GPR address relative to the base address (CP), while 'Rb' specifies a 4 bit byte GPR address relative to the base address (CP).
- **reg:** Specifies direct access to any (E)SFR or GPR in the currently active context (register bank). 'reg' requires eight bits in the instruction format. Short 'reg' addresses from 00h to EFh always specify (E)SFRs. In this case, the factor 'Δ' equals 2 and the base address is 00'F000h for the standard SFR area, or 00'FE00h for the extended ESFR area. 'reg' accesses to the ESFR area require a preceding EXT\*R instruction to switch the base address (not available in the ST10X166 devices). Depending on the opcode of an instruction, either the total word (for word operations), or the low byte (for byte operations) of an SFR can be addressed via 'reg'. Note that the high byte of an SFR cannot be accessed by the 'reg' addressing mode. Short 'reg' addresses from F0h to FFh always specify GPRs. In this case, only the lower four bits of 'reg' are significant for physical address generation, therefore it can be regarded as identical to the address generation described for the 'Rb' and 'Rw' addressing modes.
- **bitoff:** Specifies direct access to any word in the bit-addressable memory space. 'bitoff' requires eight bits in the instruction format. Depending on the specified 'bitoff' range, different base addresses are used to generate physical addresses: Short 'bitoff' addresses from 00h to 7Fh use 00'FD00h as a base address, therefore they specify the 128 highest internal RAM word locations (00'FD00h to 00'FDFEh). Short 'bitoff' addresses from 80h to EFh use 00'FF00h as a base address to specify the highest internal SFR word locations (00'FF00h to 00'FDEh) or use 00'F100h as a base address to specify the highest internal SFR word locations (00'FF00h to 00'FFDEh) or use 00'F100h to 00'F1DEh). 'bitoff' accesses to the ESFR area require a preceding EXT\*R instruction to switch the base address (not available in the ST10X166 devices). For short 'bitoff' addresses from F0h to FFh, only the lowest four bits and the contents of the CP register are used to generate the physical address of the selected word GPR.
- **bitaddr:** Any bit address is specified by a word address within the bit-addressable memory space (see 'bitoff'), and by a bit position ('bitpos') within that word. Thus, 'bitaddr' requires twelve bits in the instruction format.

<u>ل</u>حک

## 2.2 Long addressing mode

Long addressing mode uses one of the four DPP registers to specify a physical 18-bit or 24bit address. Any word or byte data within the entire address space can be accessed in this mode. All devices except the ST10X166 support an override mechanism for the DPP addressing scheme (see section 2.2.1).

Note: Word accesses on odd byte addresses are not executed, but rather trigger a hardware trap. After reset, the DPP registers are initialized so that all long addresses are directly mapped onto the identical physical addresses, within segment 0.

Long addresses (16-bit) are treated in two parts. Bits 13...0 specify a 14-bit data page offset, and bits 15...14 specify the Data Page Pointer (1 of 4). The DPP is used to generate the physical 18-bit or 24-bit address (see figure below).

Figure 2.1 Interpretation of a 16-bit long address



All ST10 devices (with the exception of the ST10X166) support an address space of up to 16 MByte, so only the lower ten bits (4 in the case of the ST10X166) of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical address. The ST10X166 supports an address space of up to 256KBytes, so only the lower 4 bits of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical bits of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical bits of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical address.

The long addressing mode is referred to by the mnemonic 'mem'.

Figure 2.2	Summary of long address modes
------------	-------------------------------

Mnemonic	Physical Address	Long Address Range	Scope of Access
mem	(DPP0) ∥ mem∧3FFFh	0000h3FFFh	Any Word or Byte
	(DPP1)      mem∧3FFFh	4000h7FFFh	
	(DPP2)    mem∧3FFFh	8000hBFFFh	
	(DPP3)    mem∧3FFFh	C000hFFFFh	
mem	pag ∥ mem∧3FFFh	0000hFFFFh (14-bit)	Any Word or Byte
mem	seg    mem	0000hFFFFh (16-bit)	Any Word or Byte



#### 2.2.1 DPP override mechanism (not available for ST10X166 devices)

The DPP override mechanism temporarily bypasses the DPP addressing scheme.

The EXTP(R) and EXTS(R) instructions override this addressing mechanism. Instruction EXTP(R) replaces the content of the respective DPP register, while instruction EXTS(R) concatenates the complete 16-bit long address with the specified segment base address. The overriding page or segment may be specified directly as a constant (#pag, #seg) or by a word GPR (Rw).



Figure 2.3 Overriding the DPP mechanism



## 2.3 Indirect addressing modes

Indirect addressing modes can be considered as a combination of short and long addressing modes. In this mode, long 16-bit addresses are specified indirectly by the contents of a word GPR, which is specified directly by a short 4-bit address ('Rw'=0 to 15). Some indirect addressing modes add a constant value to the GPR contents before the long 16-bit address is calculated. Other indirect addressing modes allow decrementing or incrementing of the indirect address pointers (GPR content) by 2 or 1 (referring to words or bytes).

In each case, one of the four DPP registers is used to specify the physical 18-bit or 24-bit addresses. Any word or byte data within the entire memory space can be addressed indirectly. Note that EXTP(R) and EXTS(R) instructions override the DPP mechanism.

Instructions using the lowest four word GPRs (R3...R0) as indirect address pointers are specified by short 2-bit addresses.

Word accesses on odd byte addresses are not executed, but rather trigger a hardware trap. After reset, the DPP registers are initialized in a way that all indirect long addresses are directly mapped onto the identical physical addresses.

Physical addresses are generated from indirect address pointers by the following algorithm:

Step 1: Calculate the physical address of the word GPR which is used as indirect address pointer, by using the specified short address ('Rw') and the current register bank base address (CP).

#### **GPR Address = (CP) + 2** \* **Short Address -** $\Delta$ ; [optional step!]

Step 2: Pre-decremented indirect address pointers ('-Rw') are decremented by a datatype-dependent value (Δ=1 for byte operations, Δ=2 for word operations), before the long 16-bit address is generated:

#### (GPR Address) = (GPR Address) - $\Delta$ ; [optional step!]

Step 3: Calculate the long 16-bit address by adding a constant value (if selected) to the content of the indirect address pointer:

#### Long Address = (GPR Pointer) + Constant

Step 4: Calculate the physical 18-bit or 24-bit address using the resulting long address and the corresponding DPP register content (see long 'mem' addressing modes).

#### Physical Address = (DPPi) + Page offset

Step 5: Post-Incremented indirect address pointers ('Rw+') are incremented by a datatype-dependent value ( $\Delta$ =1 for byte operations,  $\Delta$ =2 for word operations):

#### (GPR Pointer) = (GPR Pointer) + $\Delta$ ; [optional step!]

The following indirect addressing modes are provided:



#### **Addressing Modes**

Mnemonic	Notes
[Rw]	Most instructions accept any GPR (R15R0) as indirect address pointer.
	Some instructions, however, only accept the lower four GPRs (R3R0).
[Rw+]	The specified indirect address pointer is automatically post-incremented by 2 or 1 (for
	word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically pre-decremented by 2 or 1 (for
	word or byte data operations) before the access.
[Rw+#data16]	A

Table 2.2 Table of indirect address modes

## 2.4 Constants

The ST10 Family instruction set supports the use of wordwide or bytewide immediate constants. For optimum utilization of the available code storage, these constants are represented in the instruction formats by either 3, 4, 8 or 16 bits. Therefore, short constants are always zero-extended while long constants are truncated if necessary to match the data format required for the particular operation (see table below):

Table 2.3Table of constants

Mnemonic	Word Operation	Byte Operation
#data3	0000h + data3	00h + data3
#data4	0000h + data4	00h + data4
#data8	0000h + data8	data8
#data16	data16	data16 ∧ FFh
#mask	0000h + mask	mask

Note: Immediate constants are always signified by a leading number sign '#'.



# 2.5 Branch target addressing modes

Jump and Call instructions use different addressing modes to specify the target address and segment. Relative, absolute and indirect modes can be used to update the Instruction Pointer register (IP), while the Code Segment Pointer register (CSP) can only be updated with an absolute value. A special mode is provided to address the interrupt and trap jump vector table situated in the lowest portion of code segment 0.

Mnemonic Target Address		Target Segment	Valid Address Range			
caddr	(IP) = caddr	-	caddr = 0000hFFFEh			
rel	(IP) = (IP) + 2*rel	-	rel = 00h7Fh			
	$(IP) = (IP) + 2^{*}(\sim rel+1)$	-	rel = 80hFFh			
[Rw]	$(IP) = ((CP) + 2^*Rw)$	-	Rw = 015			
seg	-	(CSP) = seg	seg = 03			
#trap7	(IP) = 0000h + 4*trap7	(CSP) = 0000h	trap7 = 00h7Fh			

 Table 2.4
 Branch target address summary

- **caddr:** Specifies an absolute 16-bit code address within the current segment. Branches MAY NOT be taken to odd code addresses. Therefore, the least significant bit of 'caddr' must always contain a '0', otherwise a hardware trap would occur.
- **rel:** Represents an 8-bit signed word offset address relative to the current Instruction Pointer contents which points to the instruction after the branch instruction. Depending on the offset address range, either forward ('rel'= 00h to 7Fh) or backward ('rel'= 80h to FFh) branches are possible. The branch instruction itself is repeatedly executed, when 'rel' = '-1' (FF<sub>h</sub>) for a word-sized branch instruction, or 'rel' = '-2' (FEh) for a double-word-sized branch instruction.
- [Rw]: The 16-bit branch target instruction address is determined indirectly by the content of a word GPR. In contrast to indirect data addresses, indirectly specified code addresses are NOT calculated by additional pointer registers (e.g. DPP registers). Branches MAY NOT be taken to odd code addresses. Therefore, to prevent a hardware trap, the least significant bit of the address pointer GPR must always contain a '0.
- seg: Specifies an absolute code segment number. All devices (except the ST10X166) support 256 different code segments, so only the eight lower bits of the 'seg' operand value are used for updating the CSP register. The ST10X166 supports 4 different code segments so only the two lower bits of the 'seg' operand value are used for updating the CSP register
- **#trap7:** Specifies a particular interrupt or trap number for branching to the corresponding interrupt or trap service routine by a jump vector table. Trap numbers from 00h to 7Fh can be specified, which allows access to any double word code location within the address range 00'0000h...00'01FCh in code segment 0 (i.e. the interrupt jump vector table). For further information on the relation between trap numbers and interrupt or trap sources, refer to the device user manual section on "Interrupt and Trap Functions".

# **3** Instruction Execution Times

The instruction execution time depends on where the instruction is fetched from, and where the operands are read from or written to. The fastest processing mode is to execute a program fetched from the internal ROM. In this case most of the instructions can be processed in just one machine cycle.

All external memory accesses are performed by the on-chip External Bus Controller (EBC) which works in parallel with the CPU. Instructions from external memory cannot be processed as fast as instructions from the internal ROM, because it is necessary to perform data transfers sequentially via the external interface. In contrast to internal ROM program execution, the time required to process an external program additionally depends on the length of the instructions and operands, on the selected bus mode, and on the duration of an external memory cycle.

Processing a program from the internal RAM space is not as fast as execution from the internal ROM area, but it is flexible (i.e. for loading temporary programs into the internal RAM via the chip's serial interface, or end-of-line programming via the bootstrap loader).

The following description evaluates the minimum and maximum program execution times. which is sufficient for most requirements. For an exact determination of the instructions' state times, the facilities provided by simulators or emulators should be used.

This section defines measurement units, summarizes the minimum (standard) state times of the 16-bit microcontroller instructions, and describes the exceptions from the standard timing.

## 3.1 Definition of measurement units

The following measurement units are used to define instruction processing times:

- [f<sub>CPU</sub>]: CPU operating frequency (may vary from 1 MHz to 20 MHz).
- **[State]**: One state time is specified by one CPU clock period. Therefore, one State is used as the basic time unit, because it represents the shortest period of time which has to be considered for instruction timing evaluations.

1 [State]	$= 1/f_{CPU}$	[s]	; for <i>f<sub>CPU</sub></i> = variable
	= 50	[ns]	; for <i>f</i> <sub>CPU</sub> = 20 MHz

**[ACT]:** This ALE (Address Latch Enable) Cycle Time specifies the time required to perform one external memory access. One ALE Cycle Time consists of either two (for demultiplexed external bus modes) or three (for multiplexed external bus modes) state times plus a number of state times, which is determined by the number of waitstates programmed in the MCTC (Memory Cycle Time Control) and MTTC (Memory Tristate Time Control) bit fields of the SYSCON/BUSCONx registers.

For demultiplexed external bus modes: 1 ACT = (2 + (15 - MCTC) + (1 - MTTC)) States  $= 100 \text{ ns} \dots 900 \text{ ns}$ ; for  $f_{CPU} = 20 \text{ MHz}$ For multiplexed external bus modes: 1 ACT = 3 + (15 - MCTC) + (1 - MTTC) States $= 150 \text{ ns} \dots 950 \text{ ns}$ ; for  $f_{CPU} = 20 \text{ MHz}$ 

 $T_{tot}$  The total time ( $T_{tot}$ ) taken to process a particular part of a program can be calculated by the sum of the single instruction processing times ( $T_{ln}$ ) of the considered instructions plus an offset value of 6 state times which takes into account the solitary filling of the pipeline, as follows:

 $T_{tot} = T_{11} + T_{12} + ... + T_{1n} + 6 \cdot States$ 

**T**<sub>In</sub> The time ( $T_{In}$ ) taken to process a single instruction, consists of a minimum number ( $T_{Imin}$ ) plus an additional number ( $T_{Iadd}$ ) of instruction state times and/or ALE Cycle Times, as follows:

 $T_{In} = T_{Imin} + T_{Iadd}$ 

## 3.2 Minimum state times

The table below shows the minimum number of state times required to process an instruction fetched from the internal ROM ( $T_{\text{Imin}}$  (ROM)). This table can also be used to calculate the minimum number of state times for instructions fetched from the internal RAM ( $T_{\text{Imin}}$  (RAM)), or ALE Cycle Times for instructions fetched from the external memory ( $T_{\text{Imin}}$  (ext)).

Most of the 16-bit microcontroller instructions (except some branch, multiplication, division and a special move instructions) require a minimum of two state times. For internal ROM program execution, execution time has no dependence on instruction length, except for some special branch situations.

To evaluate the execution time for the injected target instruction of a cache jump instruction, it can be considered as if it was executed from the internal ROM, regardless of which memory area the rest of the current program is really fetched from.

For some of the branch instructions the table below represents both the standard number of state times (i.e. the corresponding branch is taken) and an additional  $T_{\text{Imin}}$  value in parentheses, which refers to the case where, either the branch condition is not met, or a cache jump is taken.

Instructions executed from the internal RAM require the same minimum time as they would if they were fetched from the internal ROM, plus an instruction-length dependent number of state times, as follows:

15/185

Instruction	T <sub>lmin</sub> [Sta	7 <sub>Imin</sub> (ROM) [States]		T <sub>IMin</sub> (ROM) (@ 20 MHz CPU clock)	
CALLI, CALLA	4	(2)	200	(100)	
CALLS, CALLR, PCALL	4		200		
JB, JBC, JNB, JNBS	4	(2)	200	(100)	
JMPS	4		200		
JMPA, JMPI, JMPR	4	(2)	200	(100)	
MUL, MULU	10		500		
DIV, DIVL, DIVU, DIVLU	20		1000		
MOV[B] Rn, [Rm+#data16]	4		200		
RET, RETI, RETP, RETS	4		200		
TRAP	4		200		
All other instructions	2		100		

#### Table 3.1 Minimum instruction state times [Unit = ns]

For 2-byte instructions: T<sub>Imin</sub>(RAM) = T<sub>Imin</sub>(ROM) + 4 \* States

For 4-byte instructions:  $T_{\text{Imin}}(\text{RAM}) = T_{\text{Imin}}(\text{ROM}) + 6 \cdot \text{States}$ 

Unlike internal ROM program execution, the minimum time  $T_{\text{Imin}}(\text{ext})$  to process an external instruction also depends on instruction length.  $T_{\text{Imin}}(\text{ext})$  is either 1 ALE Cycle Time for most of the 2-byte instructions, or 2 ALE Cycle Times for most of the 4-byte instructions. The following formula represents the minimum execution time of instructions fetched from an external memory via a 16-bit wide data bus:

For 2-byte instructions: T<sub>Imin</sub>(ext) = 1\*ACT + (T<sub>Imin</sub>(ROM) - 2) \* States

For 4-byte instructions:  $T_{Imin}(ext) = 2 \cdot ACTs + (T_{Imin}(ROM) - 2) \cdot States$ 

Note: For instructions fetched from an external memory via an 8-bit wide data bus, the minimum number of required ALE Cycle Times is twice the number for those of a 16-bit wide bus.

# 3.3 Additional state times

Some operand accesses can extend the execution time of an instruction  $T_{In}$ . Since the additional time  $T_{Iadd}$  is generally caused by internal instruction pipelining, it may be possible to minimize the effect by rearranging the instruction sequences. Simulators and emulators offer a high level of programmer support for program optimization.

The following operands require additional state times:

Internal ROM operand reads: T<sub>ladd</sub> = 2 \* States

Both byte and word operand reads always require 2 additional state times.

Internal RAM operand reads via indirect addressing modes:  $T_{ladd} = 0$  or 1 \* State Reading a GPR or any other directly addressed operand within the internal RAM space does NOT cause additional state times. However, reading an indirectly addressed internal RAM

operand will extend the processing time by 1 state time, if the preceding instruction autoincrements or auto-decrements a GPR, as shown in the following example:

I <sub>n</sub>	: MOV R1, [R0+]	; auto-increment R0
I <sub>n+1</sub>	: MOV [R3], [R2]	; if R2 points into the internal RAM space: ; T <sub>ladd</sub> = 1 <sub>*</sub> State

In this case, the additional time can be avoided by putting another suitable instruction before the instruction  $I_{n+1}$  indirectly reading the internal RAM.

Internal SFR operand reads:  $T_{ladd} = 0, 1 * State or 2 * States$ 

SFR read accesses do NOT usually require additional processing time. In some rare cases, however, either one or two additional state times will be caused by particular SFR operations:

• Reading an SFR immediately after an instruction, which writes to the internal SFR space, as shown in the following example:

In	: MOV T0, #1000h	; write to Timer 0
I <sub>n+1</sub>	: ADD R3, T1	; read from Timer 1: T <sub>ladd</sub> = 1 <sub>*</sub> State

• Reading the PSW register immediately after an instruction which implicitly updates the condition flags, as shown in the following example:

I <sub>n</sub>	: ADD R0, #1000h	; implicit modification of PSW flags
I <sub>n+1</sub>	: BAND C, Z	; read from PSW: T <sub>ladd</sub> = 2 <sub>*</sub> States

• Implicitly incrementing or decrementing the SP register immediately after an instruction which explicitly writes to the SP register, as shown in the following example:

I <sub>n</sub>	: MOV	SP, #0FB00h	; explicit update of the stack pointer
I <sub>n+1</sub>	: SCX	R1, #1000h	; implicit decrement of the stack pointer:
			: T <sub>ladd</sub> = 2 * States

In each of these above cases, the extra state times can be avoided by putting other suitable instructions before the instruction  $I_{n+1}$  reading the SFR.

#### External operand reads: $T_{ladd} = 1 * ACT$

57

Any external operand reading via a 16-bit wide data bus requires one additional ALE Cycle Time. Reading word operands via an 8-bit wide data bus takes twice as much time (2 ALE Cycle Times) as the reading of byte operands.

#### External operand writes: T<sub>ladd</sub> = 0 \* State ... 1 \* ACT

Writing an external operand via a 16-bit wide data bus takes one additional ALE Cycle Time. For timing calculations of external program parts, this extra time must always be considered. The value of  $T_{ladd}$  which must be considered for timing evaluations of internal program parts, may fluctuate between 0 state times and 1 ALE Cycle Time. This is because external writes

#### **Instruction Execution Times**

are normally performed in parallel to other CPU operations. Thus,  $T_{ladd}$  could already have been considered in the standard processing time of another instruction. Writing a word operand via an 8-bit wide data bus requires twice as much time (2 ALE Cycle Times) as the writing of a byte operand.

#### Jumps into the internal ROM space: Tladd = 0 or 2 \* States

The minimum time of 4 state times for standard jumps into the internal ROM space will be extended by 2 additional state times, if the branch target instruction is a double word instruction at a non-aligned double word location (xxx2h, xxx6h, xxxAh, xxxEh), as shown in the following example:

label	:	; any non-aligned double word instruction : (e.g. at location 0FFEh)
	:	
I <sub>n+1</sub>	: JMPA cc-UC, label	; if a standard branch is taken: : T <sub>Iadd</sub> = 2 ∗ States (T <sub>In</sub> = 6 ∗ States)

A cache jump, which normally requires just 2 state times, will be extended by 2 additional state times, if both the cached jump target instruction and the following instruction are non-aligned double word instructions, as shown in the following example:

label	:	; any non-aligned double word instruction : (e.g. at location 12FAh)
I <sub>t+1</sub>	:	; any non-aligned double word instruction : (e.g. at location 12FEh)
I <sub>n+1</sub>	:JMPR cc-UC, label	; provided that a cache jump is taken: : T <sub>ladd</sub> = 2 <sub>*</sub> States (T <sub>In</sub> = 4 <sub>*</sub> States)

If necessary, these extra state times can be avoided by allocating double word jump target instructions to aligned double word addresses (xxx0h, xxx4h, xxx8h, xxxCh).

#### Testing Branch Conditions: $T_{ladd} = 0$ or 1 \* States

NO extra time is usually required for a conditional branch instructions to decide whether a branch condition is met or not. However, an additional state time is required if the preceding instruction writes to the PSW register, as shown in the following example:

I <sub>n</sub>	: BSET USR0	; write to PSW
l <sub>n+1</sub> State	:JMPR cc-Z, label	;test condition flag in PSW: $T_{ladd}$ = 1 *

In this case, the extra state time can be intercepted by putting another suitable instruction before the conditional branch instruction.

# 4 Instruction Set Summary

## 4.1 X-ref tables of opcode by mnemonic

	0x	1x	2x	3x	4x	5x	6x	7x
x0	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x1	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x2	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x3	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x4	ADD	ADDC	SUB	SUBC	-	XOR	AND	OR
x5	ADDB	ADDCB	SUBB	SUBCB	-	XORB	ANDB	ORB
x6	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x7	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x8	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x9	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
XA	BFLDL	BFLDH	BCMP	BMOVN	BMOV	BOR	BAND	BXOR
хB	MUL	MULU	PRIOR	-	DIV	DIVU	DIVL	DIVLU
xC	ROL	ROL	ROR	ROR	SHL	SHL	SHR	SHR
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
хE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
хF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

 Table 4.1
 Hexadecimal opcode by mnemonic

#### Table 4.2Hexadecimal opcode by mnemonic

	8x	9x	Ах	Bx	Сх	Dx	Ex	Fx
x0	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	MOV	MOV
x1	NEG	CPL	NEGB	CPLB	-	AT/EXTR	MOVB	MOVB
x2	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	PCALL	MOV
x3	-	-	-	-	-	-	-	MOVB
x4	MOV	MOV	MOVB	MOVB	MOV	MOV	MOVB	MOVB
x5	-	-	DISWDT	EINIT	MOVBZ	MOVBS	-	-
x6	CMPI1	CMPI2	CMPD1	CMPD2	SCXT	SCXT	MOV	MOV
x7	IDLE	PWRDN	SRVWDT	SRST	-	EXTP/S/R	MOVB	MOVB
x8	MOV	MOV	MOV	MOV	MOV	MOV	MOV	-
x9	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	-
XA	JB	JNB	JBC	JNBS	CALLA	CALLS	JMPA	JMPS
xВ	-	TRAP	CALLI	CALLR	RET	RETS	RETP	RETI
xC	-	JMPI	ASHR	ASHR	NOP	EXTP/S/R	PUSH	POP
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
хE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

# 4.2 X-ref table of mnemonic, address mode & number of bytes

Table 4.3 lists the instructions by their mnemonic and identifies the addressing modes that may be used with a specific instruction and the instruction length, depending on the selected addressing mode (in bytes).

Mnemonic	Addressing Modes	Bytes	Mnemonic	AddressingModes	Bytes
ADD[B]	Rwn Rwm <sup>1)</sup>	2	CPL[B]	Rwn <sup>1)</sup>	2
ADDC[B]	Rwn Rwi] <sup>1)</sup>	2	NEG[B]		
AND[B]	Rwn Rwi+] <sup>1)</sup>	2	DIV	Rwn	2
OR[B]	Rwn #data3 <sup>1)</sup>	2	DIVL		
SUB[B]	reg #data16 <sup>2)</sup>	4	DIVLU		
SUBC[B]	reg mem	4	DIVU		
XOR[B]	mem reg	4	MUL	RwnRwm	2
			MULU		
ASHR	Rwn Rwm	2	CMPD1/2	Rwn #data4	2
ROL/ROR	Rwn #data4	2	CMPI1/2	Rwn #data16	4
SHL / SHR				Rwn mem	4
BAND	bitaddrZ.z bitaddrQ.q	4	CMP[B]	Rwn Rwm <sup>1)</sup>	2
BCMP				Rwn [Rwi] <sup>1)</sup>	2
BMOV				Rwn [Rwi+] <sup>1)</sup>	2
BMOVN				Rwn #data3 <sup>1)</sup>	2
BOR /				reg #data16 <sup>2)</sup>	4
BXOR				reg mem	4
BCLR	bitaddrQ.q	2	CALLA	cc caddr	4
BSET			JMPA		
BFLDH	bitoffQ #mask8#data8	2	CALLI	cc [Rwn]	2
BFLDL			JMPI		

Table 4 3	Instruction	mnemonic by	v address	modes and	Inumber	of by	tes
1 abie 4.5	manuchon		y auuress	moues and	inumper		ies



Mnemonic	Addressing Modes	Bytes	Mnemonic	AddressingModes	Bytes
MOV[B]	Rwn Rwm <sup>1)</sup>	2	CALLS	segcaddr	4
	Rwn #data4 <sup>1)</sup>	2	JMPS		
	Rwn Rwm] <sup>1)</sup>	2	CALLR	rel	2
	Rwn Rwm+] <sup>1)</sup>	2	JMPR	cc rel	2
	[Rwm Rwn <sup>1)</sup>	2	JB	bitaddrQ.q rel	4
	[-Rwm] Rwn ''	2	JBC		
	[Rwn] [Rwm]	2	JNB		
	[Rwn+] [Rwm]	2	JNBS		
	[Kwn] [Kwm+]	2	PCALL	reg caddr	4
	$reg = #data16^{2}$		POP	reg	2
	$\begin{bmatrix} 1 \text{ eg} & \#\text{uatarto} \\ \text{Rwp} & [\text{Rwp} \#\text{d}161^1) \end{bmatrix}$		PUSH		
	[Rwm+#d16] Rwn <sup>1)</sup>		RETP		
	[Rwn] mem	4	SCXT	reg #data16	4
	mem [Rwn]	4		reg mem	4
	rea mem	4	PRIOR	Rwn Rwm	2
	mem reg	4			
MOVBS	Rwn Rbm	2	TRAP	#trap7	2
MOVBZ	reg mem	4	ATOMIC	#data2 <sup>3)</sup>	2
	mem reg	4	EXTR		
EXTS	Rwm #data2 <sup>3)</sup>	2	EXTP	Rwm #data2 <sup>3)</sup>	2
EXTSR	#seg #data2	4	EXTPR	#pag #data2	4
NOP	-	2	SRST/IDLE	-	4
RET			PWRDN		
RETI			SRVWDT		
RETS			DISWDT		
			EINIT		

 Table 4.3
 Instruction mnemonic by address modes and number of bytes (cont'd)

Notes 1: Byte oriented instructions (suffix 'B') use Rb instead of Rw (not with [Rwn]!).

2: Byte oriented instructions (suffix 'B') use #data8 instead of #data16.

3: The ATOMIC and EXTended instructions are not available in the ST10X166 devices.

## 4.3 Instruction set ordered by functional group

The following tables list the instruction set by functional group. Within each table the instructions are listed alphabetically:

Table 4.4 Arithmetic instructions

Table 4.5 Logical instructions

Table 4.6 Boolean bit map instructions

Table 4.7 Compare and loop instructions

Table 4.8 Prioritize instructions

Table 4.9 Shift and rotate instructions

Table 4.10 Data movement instructions

Table 4.11 Jump and Call Instructions

Table 4.12 System Stack Instructions

Table 4.13 Return Instructions

Table 4.14 System Control Instructions

Table 4.15 Miscellaneous instructions

The minimum number of state times required for instruction execution are given for the following configurations: internal ROM, internal RAM, external memory with a 16-bit demultiplexed and multiplexed bus or an 8-bit demultiplexed and multiplexed bus. These state time figures do not take into account possible wait states on external busses or possible additional state times induced by operand fetches. The following notes apply to this summary:

#### Data addressing modes

Rw:	Word GPR (R0, R1, , R15)
Rb:	Byte GPR (RL0, RH0,, RL7, RH7)
reg:	SFR or GPR (in case of a byte operation on an SFR, only the low byte can be accessed via 'reg')
mem:	Direct word or byte memory location
[]:	Indirect word or byte memory location (Any word GPR can be used as indirect address pointer, except for the arithmetic, logical and compare instructions, where only R0 to R3 are allowed)
bitaddr:	Direct bit in the bit-addressable memory area
bitoff:	Direct word in the bit-addressable memory area
#data:	Immediate constant (The number of significant bits which can be specified by the user is represented by the respective appendix 'x')
#mask8:	Immediate 8-bit mask used for bit-field modifications



#### Multiply and divide operations

The MDL and MDH registers are implicit source and/or destination operands of the multiply and divide instructions.

#### Branch target addressing modes

- caddr: Direct 16-bit jump target address (Updates the Instruction Pointer)
- seg: Direct 2-bit segment address (Updates the Code Segment Pointer)
- rel: Signed 8-bit jump target word offset address relative to the Instruction Pointer of the following instruction
- #trap7: Immediate 7-bit trap or interrupt number.

#### **Extension operations**

The EXT\* instructions override the standard DPP addressing scheme:

- #pag10: Immediate 10-bit page address.
- #seg8: Immediate 8-bit segment address.
- Note: The EXTended instructions are not available in the ST10X166 devices.

#### Branch condition codes

cc: Symbolically specifiable condition codes

cc_UC	Unconditional
cc_Z	Zero
cc_NZ	Not Zero
cc_V	Overflow
cc_NV	No Overflow
cc_N	Negative
cc_NN	Not Negative
cc_C	Carry
cc_NC	No Carry
cc_EQ	Equal
cc_NE	Not Equal
cc_ULT	Unsigned Less Than
cc_ULE	Unsigned Less Than or Equal
cc_UGE	Unsigned Greater Than or Equal
cc_UGT	Unsigned Greater Than
cc_SLE	Signed Less Than or Equal
cc_SGE	Signed Greater Than or Equal
cc_SGT	Signed Greater Than
cc_NET	Not Equal and Not End-of-Table

Note: <sup>)</sup> The EXTended instructions are not available in the ST10X166 devices.



Table 4.4	Arithmetic	instructions
-----------	------------	--------------

Mnemonic	Description	Int.ROM	Int.RAM	16-bit Non -Mux	16-bit Mux	8-bitNon -Mux	8-bit Mux	Bytes
ADD Rw, Rw	Add direct word GPR to direct GPR	2	6	2	3	4	6	2
ADD Rw, [Rw]	Add indirect word memory to direct GPR	2	6	2	3	4	6	2
ADD Rw, [Rw +]	Add indirect word memory to direct GPR	2	6	2	3	4	6	2
	and post- increment source pointer by 2							
ADD Rw, #data3	Add immediate word data to direct GPR	2	6	2	3	4	6	2
ADD reg, #data16	Add immediate word data to direct register	2	8	4	6	8	12	4
ADD reg, mem	Add direct word memory to direct register	2	8	4	6	8	12	4
ADD mem, reg	Add direct word register to direct memory	2	8	4	6	8	12	4
ADDB Rb, Rb	Add direct byte GPR to direct GPR	2	6	2	3	4	6	2
ADDB Rb, [Rw]	Add indirect byte memory to direct GPR	2	6	2	3	4	6	2
ADDBRb, [Rw +]	Add indirect byte memory to direct GPR	2	6	2	3	4	6	2
	and post-increment source pointer by 1							
ADDB Rb, #data3	Add immediate byte data to direct GPR	2	6	2	3	4	6	2
ADDB reg, #data16	Add immediate byte data to direct register	2	8	4	6	8	12	4
ADDB reg, mem	Add direct byte memory to direct register	2	8	4	6	8	12	4
ADDB mem, reg	Add direct byte register to direct memory	2	8	4	6	8	12	4
ADDCRw, Rw	Add direct word GPR to direct GPR with	2	6	2	3	4	6	2
	Carry							
ADDCRw, [Rw]	Add indirect word memory to direct GPR	2	6	2	3	4	6	2
	with Carry							
ADDCRw, [Rw +]	Add indirect word memory to direct GPR	2	6	2	3	4	6	2
	with Carry and post-increment source							
	pointer by 2							
ADDCRw, #data3	Add immediate word data to direct GPR with Carry	2	6	2	3	4	6	2
ADDCreg, #data16	Add immediate word data to direct register	2	8	4	6	8	12	4
	with Carry							
ADDC reg, mem	Add direct word memory to direct register	2	8	4	6	8	12	4
	with Carry							
ADDC mem, reg	Add direct word register to direct memory	2	8	4	6	8	12	4
	with Carry							
ADDCBRb, Rb	Add direct byte GPR to direct GPR with	2	6	2	3	4	6	2
	Carry							
ADDCBRb, [Rw]	Add indirect byte memory to direct GPR	2	6	2	3	4	6	2
	with Carry							
ADDCBRb, [Rw +]	Add indirect byte memory to direct GPR	2	6	2	3	4	6	2
	with Carry and post-increment source							
	pointer by 1							



Mnemonic	Description	Int.ROM	Int.RAM	16-bit Non -Mux	16-bit Mux	8-bitNon -Mux	8-bit Mux	Bytes
ADDCBRb, #data3	Add immediate byte data to direct GPR with Carry	2	6	2	3	4	6	2
ADDCBreg, #data16	Add immediate byte data to direct register with Carry	2	8	4	6	8	12	4
ADDCBreg, mem	Add direct byte memory to direct register with Carry	2	8	4	6	8	12	4
ADDCBmem, reg	Add direct byte register to direct memory with Carry	2	8	4	6	8	12	4
CPL Rw	Complement direct word GPR	2	6	2	3	4	6	2
CPLB Rb	Complement direct byte GPR	2	6	2	3	4	6	2
DIV Rw	Signed divide register MDL by direct GPR (16-/16-bit)	20	24	20	21	22	24	2
DIVL Rw	Signed long divide register MD by direct GPR (32-/16-bit)	20	24	20	21	22	24	2
DIVLURw	Unsigned long divide register MD by direct GPR (32-/16-bit)	20	24	20	21	22	24	2
DIVU Rw	Unsigned divide register MDL by direct GPR (16-/16-bit)	20	24	20	21	22	24	2
MUL Rw, Rw	Signed multiply direct GPR by direct GPR (16-16-bit)	10	14	10	11	12	14	2
MULURw, Rw	Unsigned multiply direct GPR by direct GPR (16-16-bit)	10	14	10	11	12	14	2
NEG Rw	Negate direct word GPR	2	6	2	3	4	6	2
NEGBRb	Negate direct byte GPR	2	6	2	3	4	6	2
SUB Rw, Rw	Subtract direct word GPR from direct GPR	2	6	2	3	4	6	2
SUB Rw, [Rw]	Subtract indirect word memory from direct GPR	2	6	2	3	4	6	2
SUB Rw, [Rw +]	Subtract indirect word memory from direct GPR & post-increment source pointer by 2	2	6	2	3	4	6	2
SUB Rw, #data3	Subtract immediate word data from direct GPR	2	6	2	3	4	6	2
SUB reg, #data16	Subtract immediate word data from direct register	2	8	4	6	8	12	4
SUB reg, mem	Subtract direct word memory from direct register	2	8	4	6	8	12	4
SUB mem, reg	Subtract direct word register from direct memory	2	8	4	6	8	12	4
SUBB Rb, Rb	Subtract direct byte GPR from direct GPR	2	6	2	3	4	6	2

 Table 4.4
 Arithmetic instructions (cont'd)

#### Instruction Set Summary

Table 4.4	Arithmetic instructions	(cont'd)
-----------	-------------------------	----------

Mnemonic	Description	Int.ROM	Int.RAM	16-bit Non -Mux	16-bit Mux	8-bitNon -Mux	8-bit Mux	Bytes
SUBB Rb, [Rw]	Subtract indirect byte memory from direct GPR	2	6	2	3	4	6	2
SUBB Rb, [Rw +]	Subtract indirect byte memory from direct GPR & post-increment source pointer by 1	2	6	2	3	4	6	2
SUBB Rb, #data3	Subtract immediate byte data from direct GPR	2	6	2	3	4	6	2
SUBB reg, #data16	Subtract immediate byte data from direct register	2	8	4	6	8	12	4
SUBB reg, mem	Subtract direct byte memory from direct register	2	8	4	6	8	12	4
SUBB mem, reg	Subtract direct byte register from direct memory	2	8	4	6	8	12	4
SUBC Rw, Rw	Subtract direct word GPR from direct GPR with Carry	2	6	2	3	4	6	2
SUBC Rw, [Rw]	Subtract indirect word memory from direct GPR with Carry	2	6	2	3	4	6	2
SUBC Rw, [Rw +]	Subtract indirect word memory from direct GPR with Carry and post-increment source pointer by 2	2	6	2	3	4	6	2
SUBC Rw, #data3	Subtract immediate word data from direct GPR with Carry	2	6	2	3	4	6	2
SUBC reg, #data16	Subtract immediate word data from direct register with Carry	2	8	4	6	8	12	4
SUBC reg, mem	Subtract direct word memory from direct register with Carry	2	8	4	6	8	12	4
SUBC mem, reg	Subtract direct word register from direct memory with Carry	2	8	4	6	8	12	4
SUBCBRb, Rb	Subtract direct byte GPR from direct GPR with Carry	2	6	2	3	4	6	2
SUBCBRb, [Rw]	Subtract indirect byte memory from direct GPR with Carry	2	6	2	3	4	6	2
SUBCBRb, [Rw +]	Subtract indirect byte memory from direct GPR with Carry and post-increment source pointer by 1	2	6	2	3	4	6	2
SUBCBRb, #data3	Subtract immediate byte data from direct GPR with Carry	2	6	2	3	4	6	2
SUBCBreg, #data16	Subtract immediate byte data from direct register with Carry	2	8	4	6	8	12	4



Table 4.4 A	rithmetic instructions	(cont'd)
-------------	------------------------	----------

Mnemonic	Description	Int.ROM	Int.RAM	16-bit Non -Mux	16-bit Mux	8-bitNon -Mux	8-bit Mux	Bytes
SUBCBreg, mem	Subtract direct byte memory from direct	2	8	4	6	8	12	4
	register with Carry							
SUBCBmem, reg	Subtract direct byte register from direct	2	8	4	6	8	12	4
	memory with Carry							

#### Table 4.5Logical instructions

Mnemonic	Description	Int ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
AND Rw, Rw	Bitwise AND direct word GPR with direct GPR	2	6	2	3	4	6	2
AND Rw, [Rw]	Bitwise AND indirect word memory with direct GPR	2	6	2	3	4	6	2
AND Rw, [Rw +]	Bitwise AND indirect word memory with direct GPR and post-increment source pointer by 2	2	6	2	3	4	6	2
AND Rw, #data3	Bitwise AND immediate word data with direct GPR	2	6	2	3	4	6	2
AND reg, #data16	Bitwise AND immediate word data with direct register	2	8	4	6	8	12	4
AND reg, mem	Bitwise AND direct word memory with di- rect register	2	8	4	6	8	12	4
AND mem, reg	Bitwise AND direct word register with di- rect memory	2	8	4	6	8	12	4
ANDB Rb, Rb	Bitwise AND direct byte GPR with direct GPR	2	6	2	3	4	6	2
ANDB Rb, [Rw]	Bitwise AND indirect byte memory with direct GPR	2	6	2	3	4	6	2
ANDB Rb, [Rw +]	Bitwise AND indirect byte memory with direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
ANDB Rb, #data3	Bitwise AND immediate byte data with direct GPR	2	6	2	3	4	6	2
ANDB reg, #data16	Bitwise AND immediate byte data with direct register	2	8	4	6	8	12	4
ANDB reg, mem	Bitwise AND direct byte memory with di- rect register	2	8	4	6	8	12	4
ANDB mem, reg	Bitwise AND direct byte register with di- rect memory	2	8	4	6	8	12	4

	Mnemonic	Description	Int ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
OR	Rw, Rw	Bitwise OR direct word GPR with direct GPR	2	6	2	3	4	6	2
OR	Rw, [Rw]	Bitwise OR indirect word memory with direct GPR	2	6	2	3	4	6	2
OR	Rw, [Rw +]	Bitwise OR indirect word memory with direct GPR and post-increment source pointer by 2	2	6	2	3	4	6	2
OR	Rw, #data3	Bitwise OR immediate word data with direct GPR	2	6	2	3	4	6	2
OR	reg, #data16	Bitwise OR immediate word data with di- rect register	2	8	4	6	8	12	4
OR	reg, mem	Bitwise OR direct word memory with di- rect register	2	8	4	6	8	12	4
OR	mem, reg	Bitwise OR direct word register with di- rect memory	2	8	4	6	8	12	4
ORB	Rb, Rb	Bitwise OR direct byte GPR with direct GPR	2	6	2	3	4	6	2
ORB	Rb, [Rw]	Bitwise OR indirect byte memory with di- rect GPR	2	6	2	3	4	6	2
ORB	Rb, [Rw +]	Bitwise OR indirect byte memory with di- rect GPR andpost-increment source pointer by 1	2	6	2	3	4	6	2
ORB	Rb, #data3	Bitwise OR immediate byte data with direct GPR	2	6	2	3	4	6	2
ORB	reg, #data16	Bitwise OR immediate byte data with di- rect register	2	8	4	6	8	12	4
ORB	reg, mem	Bitwise OR direct byte memory with di- rect register	2	8	4	6	8	12	4
ORB	mem, reg	Bitwise OR direct byte register with di- rect memory	2	8	4	6	8	12	4
XOR	Rw, Rw	Bitwise XOR direct word GPR with direct GPR	2	6	2	3	4	6	2
XOR	Rw, [Rw]	Bitwise XOR indirect word memory with direct GPR	2	6	2	3	4	6	2
XOR	Rw, [Rw +]	Bitwise XOR indirect word memory with direct GPR and post-increment source pointer by 2	2	6	2	3	4	6	2
XOR	Rw, #data3	Bitwise XOR immediate word data with direct GPR	2	6	2	3	4	6	2

 Table 4.5
 Logical instructions (cont'd)



Mnemonic	Description	Int ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
XOR reg, #data16	Bitwise XOR immediate word data with direct register	2	8	4	6	8	12	4
XOR reg, mem	Bitwise XOR direct word memory with direct register	2	8	4	6	8	12	4
XOR mem, reg	Bitwise XOR direct word register with di- rect memory	2	8	4	6	8	12	4
XORBRb, Rb	Bitwise XOR direct byte GPR with direct GPR	2	6	2	3	4	6	2
XORBRb, [Rw]	Bitwise XOR indirect byte memory with direct GPR	2	6	2	3	4	6	2
XORBRb, [Rw +]	Bitwise XOR indirect byte memory with direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
XORBRb, #data3	Bitwise XOR immediate byte data with direct GPR	2	6	2	3	4	6	2
XORBreg, #data16	Bitwise XOR immediate byte data with direct register	2	8	4	6	8	12	4
XORBreg, mem	Bitwise XOR direct byte memory with di- rect register	2	8	4	6	8	12	4
XORBmem, reg	Bitwise XOR direct byte register with di- rect memory	2	8	4	6	8	12	4

Table 4.5	Logical	instructions	(cont'd)
-----------	---------	--------------	----------

#### Table 4.6 Boolean bit map instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
BAND	AND direct bit with direct bit	2	8	4	6	8	12	4
bitaddr, bitaddr								
BCLR bitaddr	Clear direct bit	2	6	2	3	4	6	2
BCMP	Compare direct bit to direct bit	2	8	4	6	8	12	4
bitaddr, bitaddr								
BFLDH	Bitwise modify masked high byte of bit-	2	8	4	6	8	12	4
bitoff, #mask8,#data8	addressable direct word memory with							
	immediate data							
BFLDL	Bitwise modify masked low byte of bit-	2	8	4	6	8	12	4
bitoff, #mask8, #data8	addressable direct word memory with							
	immediate data							
BMOV	Move direct bit to direct bit	2	8	4	6	8	12	4
bitaddr, bitaddr								

#### Instruction Set Summary

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
BMOVN biteddr, biteddr	Move negated direct bit to direct bit	2	8	4	6	8	12	4
	OD direct hit with direct hit			4		0	10	4
bitaddr. bitaddr	OR direct bit with direct bit	2	8	4	6	8	12	4
BSET bitaddr	Set direct bit	2	6	2	3	4	6	2
BXOR	XOR direct bit with direct bit	2	8	4	6	8	12	4
bitaddr, bitaddr		_				Ũ		•
CMP Rw, Rw	Compare direct word GPR to direct GPR	2	6	2	3	4	6	2
CMP Rw, [Rw]	Compare indirect word memory to direct GPR	2	6	2	3	4	6	2
CMP Rw, [Rw +]	Compare indirect word memory to direct GPR and post-increment source pointer by 2	2	6	2	3	4	6	2
CMP Rw, #data3	Compare immediate word data to direct GPR	2	6	2	3	4	6	2
CMP reg, #data16	Compare immediate word data to direct register	2	8	4	6	8	12	4
CMP reg, mem	Compare direct word memory to direct register	2	8	4	6	8	12	4
CMPBRb, Rb	Compare direct byte GPR to direct GPR	2	6	2	3	4	6	2
CMPBRb, [Rw]	Compare indirect byte memory to direct GPR	2	6	2	3	4	6	2
CMPBRb, [Rw +]	Compare indirect byte memory to direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
CMPBRb, #data3	Compare immediate byte data to direct GPR	2	6	2	3	4	6	2
CMPBreg, #data16	Compare immediate byte data to direct register	2	8	4	6	8	12	4
CMPBreg, mem	Compare direct byte memory to direct register	2	8	4	6	8	12	4

#### Table 4.6Boolean bit map instructions (cont'd)



Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
CMPD1Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 1	2	6	2	3	4	6	2
CMPD1Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 1	2	8	4	6	8	12	4
CMPD1Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 1	2	8	4	6	8	12	4
CMPD2 Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 2	2	6	2	3	4	6	2
CMPD2 Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 2	2	8	4	6	8	12	4
CMPD2Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 2	2	8	4	6	8	12	4
CMPI1Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 1	2	6	2	3	4	6	2
CMPI1Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 1	2	8	4	6	8	12	4
CMPI1Rw, mem	Compare direct word memory to direct GPR and increment GPR by 1	2	8	4	6	8	12	4
CMPI2Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 2	2	6	2	3	4	6	2
CMPI2Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 2	2	8	4	6	8	12	4
CMPI2Rw, mem	Compare direct word memory to direct GPR and increment GPR by 2	2	8	4	6	8	12	4

 Table 4.7
 Compare and loop instructions

#### Table 4.8Prioritize instructions

57

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
PRIORRw, Rw	Determine number of shift cycles to nor-	2	6	2	3	4	6	2
	malize direct word GPR and store result							
	in direct word GPR							

31/185

#### Instruction Set Summary

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
ASHR Rw, Rw	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by direct GPR	2	6	2	3	4	6	2
ASHR Rw, #data4	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2
ROL Rw, Rw	Rotate left direct word GPR; number of shift cycles specified by direct GPR	2	6	2	3	4	6	2
ROL Rw, #data4	Rotate left direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2
ROR Rw, Rw	Rotate right direct word GPR; number of shift cycles specified by direct GPR	2	6	2	3	4	6	2
ROR Rw, #data4	Rotate right direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2
SHL Rw, Rw	Shift left direct word GPR; number of shift cycles specified by direct GPR	2	6	2	3	4	6	2
SHL Rw, #data4	Shift left direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2
SHR Rw, Rw	Shift right direct word GPR; number of shift cycles specified by direct GPR	2	6	2	3	4	6	2
SHR Rw, #data4	Shift right direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2

\_\_\_\_

Table 4.9Shift and rotate instructions

#### Table 4.10Data movement instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
MOV Rw, Rw	Move direct word GPR to direct GPR	2	6	2	3	4	6	2
MOV Rw, #data4	Move immediate word data to direct GPR	2	6	2	3	4	6	2
MOV reg, #data16	Move immediate word data to direct register	2	8	4	6	8	12	4
MOV Rw, [Rw]	Move indirect word memory to direct GPR	2	6	2	3	4	6	2
MOV Rw, [Rw +]	Move indirect word memory to direct GPR and post-increment source pointer by 2	2	6	2	3	4	6	2
MOV [Rw], Rw	Move direct word GPR to indirect memory	2	6	2	3	4	6	2
MOV [-RW], Rw	Pre-decrement destination pointer by 2 and move direct word GPR to indirect memory	2	6	2	3	4	6	2
MOV [RW], [RW]	Move indirect word memory to indirect memo- ry	2	6	2	3	4	6	2

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
MOV [Rw +], [Rw]	Move indirect word memory to indirect memo- ry and post-increment destination pointer by 2	2	6	2	3	4	6	2
MOV [Rw], [Rw +]	Move indirect word memory to indirect memo- ry and post-increment source pointer by 2	2	6	2	3	4	6	2
MOV Rw, [Rw + #data16]	Move indirect word memory by base plus con- stant to direct GPR	4	10	6	8	10	14	4
MOV [Rw+#data16], Rw	Move direct word GPR to indirect memory by base plus constant	2	8	4	6	8	12	4
MOV [Rw], mem	Move direct word memory to indirect memory	2	8	4	6	8	12	4
MOV mem, [Rw]	Move indirect word memory to direct memory	2	8	4	6	8	12	4
MOV reg, mem	Move direct word memory to direct register	2	8	4	6	8	12	4
MOV mem, reg	Move direct word register to direct memory	2	8	4	6	8	12	4
MOVBRb, Rb	Move direct byte GPR to direct GPR	2	6	2	3	4	6	2
MOVBRb, #data4	Move immediate byte data to direct GPR	2	6	2	3	4	6	2
MOVBreg, #data16	Move immediate byte data to direct register	2	8	4	6	8	12	4
MOVBRb, [Rw]	Move indirect byte memory to direct GPR	2	6	2	3	4	6	2
MOVBRb, [Rw +]	Move indirect byte memory to direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
MOVB[Rw], Rb	Move direct byte GPR to indirect memory	2	6	2	3	4	6	2
MOVB[-Rw], Rb	Pre-decrement destination pointer by 1 and move direct byte GPR to indirect memory	2	6	2	3	4	6	2
MOVB[Rw], [Rw]	Move indirect byte memory to indirect memory	2	6	2	3	4	6	2
MOVB[Rw +], [Rw]	Move indirect byte memory to indirect memory and post-increment destination pointer by 1	2	6	2	3	4	6	2
MOVB[Rw], [Rw +]	Move indirect byte memory to indirect memory and post-increment source pointer by 1	2	6	2	3	4	6	2
MOVB Rb, [Rw + #data16]	Move indirect byte memory by base plus con- stant to direct GPR	4	10	6	8	10	14	4
MOVB	Move direct byte GPR to indirect memory by	2	8	4	6	8	12	4
[Rw + #data16], Rb	base plus constant							
MOVB[Rw], mem	Move direct byte memory to indirect memory	2	8	4	6	8	12	4
MOVBmem, [Rw]	Move indirect byte memory to direct memory	2	8	4	6	8	12	4
MOVBreg, mem	Move direct byte memory to direct register	2	8	4	6	8	12	4
MOVBmem, reg	Move direct byte register to direct memory	2	8	4	6	8	12	4
MOVBSRw, Rb	Move direct byte GPR with sign extension to direct word GPR	2	6	2	3	4	6	2
MOVBSreg, mem	Move direct byte memory with sign extension to direct word register	2	8	4	6	8	12	4

 Table 4.10
 Data movement instructions (cont'd)

Table 4.10	Data movement instructions	(cont'd)
------------	----------------------------	----------

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
MOVBSmem, reg	Move direct byte register with sign extension to direct word memory	2	8	4	6	8	12	4
MOVBZRw, Rb	Move direct byte GPR with zero extension to direct word GPR	2	6	2	3	4	6	2
MOVBZreg, mem	Move direct byte memory with zero extension to direct word register	2	8	4	6	8	12	4
MOVBZmem, reg	Move direct byte register with zero extension to direct word memory	2	8	4	6	8	12	4

## Table 4.11 Jump and Call Instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
CALLAcc, caddr	Call absolute subroutine if condition is met	4/2	10/	6/4	8/6	10/	14/	4
		4/0	0	4/2	= /0	0	12	
CALLICC, [RW]	Call indirect subroutine if condition is met	4/2	8/6	4/2	5/3	6/4	8/6	2
CALLRrel	Call relative subroutine	4	8	4	5	6	8	2
CALLSseg, caddr	Call absolute subroutine in any code seg-	4	10	6	8	10	14	4
	ment							
JB bitaddr, rel	Jump relative if direct bit is set	4	10	6	8	10	14	4
JBC bitaddr, rel	Jump relative and clear bit if direct bit is set	4	10	6	8	10	14	4
JMPA cc, caddr	Jump absolute if condition is met	4/2	10/	6/4	8/6	10/	14/	4
			8			8	12	
JMPI cc, [Rw]	Jump indirect if condition is met	4/2	8/6	4/2	5/3	6/4	8/6	2
JMPR cc, rel	Jump relative if condition is met	4/2	8/6	4/2	5/3	6/4	8/6	2
JMPS seg, caddr	Jump absolute to a code segment	4	10	6	8	10	14	4
JNB bitaddr, rel	Jump relative if direct bit is not set	4	10	6	8	10	14	4
JNBS bitaddr, rel	Jump relative and set bit if direct bit is not	4	10	6	8	10	14	4
	set							
PCALLreg, caddr	Push direct word register onto system	4	10	6	8	10	14	4
	stack and call absolute subroutine							
TRAP #trap7	Call interrupt service routine via immediate	4	8	4	5	6	8	2
	trap number							



Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
POP reg	Pop direct word register from system stack	2	6	2	3	4	6	2
PUSH reg	Push direct word register onto system stack	2	6	2	3	4	6	2
SCXT reg, #data16	Push direct word register onto system stack and update register with immedi- ate data	2	8	4	6	8	12	4
SCXT reg, mem	Push direct word register onto system stack and update register with direct memory	2	8	4	6	8	12	4

#### Table 4.12 System Stack Instructions

#### Table 4.13Return Instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
RET	Return from intra-segment subroutine	4	8	4	5	6	8	2
RETI	Return from interrupt service subroutine	4	8	4	5	6	8	2
RETP reg	Return from intra-segment subroutine and pop direct word register from sys- tem stack	4	8	4	5	6	8	2
RETS	Return from inter-segment subroutine	4	8	4	5	6	8	2

#### Table 4.14 System Control Instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
ATOMIC#data2	Begin ATOMIC sequence *)	2	6	2	3	4	6	2
DISWDT	Disable Watchdog Timer	2	8	4	6	8	12	4
EINIT	Signify End-of-Initialization on RSTOUT-pin	2	8	4	6	8	12	4
EXTR #data2	Begin EXTended Register sequence *)	2	6	2	3	4	6	2
EXTP Rw, #data2	Begin EXTended Page sequence <sup>*)</sup>	2	6	2	3	4	6	2
EXTP #pag10, #data2	Begin EXTended Page sequence <sup>*)</sup>	2	8	4	6	8	12	4
EXTPRRw, #data2	Begin EXTended Page and Register sequence *)	2	6	2	3	4	6	2
EXTPR #pag10, #data2	Begin EXTended Page and Register sequence *)	2	8	4	6	8	12	4
EXTS Rw, #data2	Begin EXTended Segment sequence <sup>*)</sup>	2	6	2	3	4	6	2

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
EXTS	Begin EXTended Segment sequence <sup>*)</sup>	2	8	4	6	8	12	4
#seg8, #data2								
EXTSR	Begin EXTended Segment and Register	2	6	2	3	4	6	2
Rw, #data2	sequence <sup>*)</sup>							
EXTSR	Begin EXTended Segment and Register	2	8	4	6	8	12	4
#seg8, #data2	sequence <sup>*)</sup>							
IDLE	Enter Idle Mode	2	8	4	6	8	12	4
PWRDN	Enter Power Down Mode (supposes	2	8	4	6	8	12	4
	NMI-pin being low)							
SRST	Software Reset	2	8	4	6	8	12	4
SRVWDT	Service Watchdog Timer	2	8	4	6	8	12	4

#### Table 4.14 System Control Instructions (cont'd)

### Table 4.15 Miscellaneous instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit Non-Mux	16-bit Mux	8-bit Non-Mux	8-bit Mux	Bytes
NOP	Null operation	2	6	2	3	4	6	2


# 4.4 Instruction set ordered by opcodes

The following pages list the instruction set ordered by their hexadecimal opcodes. This is used to identify specific instructions when reading executable code, i.e. during the debugging phase.

#### **Notes for Opcode Lists**

1) These instructions are encoded by means of additional bits in the operand field of the instruction

x0h – x7h:	Rw, #data3	or	Rb, #data3
x8h – xBh:	Rw, [Rw]	or	Rb, [Rw]
xCh – xFh	Rw, [Rw +]	or	Rb, [Rw +]

For these instructions only the lowest four GPRs, R0 to R3, can be used as indirect address pointers.

2) These instructions are encoded by means of additional bits in the operand field of the instruction

00xx.xxxx:	EXTS	or	ATOMIC
01xx.xxxx:	EXTP		
10xx.xxxx:	EXTSR	or	EXTR
11xx.xxxx:	EXTPR		

The ATOMIC and EXTended instructions are not available in the ST10X166 devices.

#### Notes on the JMPR instructions

The condition code to be tested for the JMPR instructions is specified by the opcode. Two mnemonic representation alternatives exist for some of the condition codes.

#### Notes on the BCLR and BSET instructions

The position of the bit to be set or to be cleared is specified by the opcode. The operand 'bitoff.n' (n = 0 to 15) refers to a particular bit within a bit-addressable word.

#### Notes on the undefined opcodes

<u>ل</u>رک

A hardware trap occurs when one of the undefined opcodes signified by '----' is decoded by The CPU.

Hex- code	Number of Bytes	Mnemonic	Operand
00	2	ADD	Rw, Rw
01	2	ADDB	Rb, Rb
02	4	ADD	reg, mem
03	4	ADDB	reg, mem
04	4	ADD	mem, reg
05	4	ADDB	mem, reg
06	4	ADD	reg, #data16
07	4	ADDB	reg, #data8
08	2	ADD	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
09	2	ADDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
0A	4	BFLDL	bitoff, #mask8, #data8
0B	2	MUL	Rw, Rw
0C	2	ROL	Rw, Rw
0D	2	JMPR	cc_UC, rel
0E	2	BCLR	bitoff.0
0F	2	BSET	bitoff.0
10	2	ADDC	Rw, Rw
11	2	ADDCB	Rb, Rb
12	4	ADDC	reg, mem
13	4	ADDCB	reg, mem
14	4	ADDC	mem, reg
15	4	ADDCB	mem, reg
16	4	ADDC	reg, #data16
17	4	ADDCB	reg, #data8
18	2	ADDC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
19	2	ADDCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
1A	4	BFLDH	bitoff, #mask8, #data8
1B	2	MULU	Rw, Rw
1C	2	ROL	Rw, #data4
1D	2	JMPR	cc_NET, rel
1E	2	BCLR	bitoff.1
1F	2	BSET	bitoff.1
20	2	SUB	Rw, Rw
21	2	SUBB	Rb, Rb
22	4	SUB	reg, mem
23	4	SUBB	reg, mem
24	4	SUB	mem, reg
25	4	SUBB	mem, reg

 Table 4.16
 Instruction set ordered by Hex code



Hex- code	Number of Bytes	Mnemonic	Operand
26	4	SUB	reg, #data16
27	4	SUBB	reg, #data8
28	2	SUB	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
29	2	SUBB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
2A	4	BCMP	bitaddr, bitaddr
2B	2	PRIOR	Rw, Rw
2C	2	ROR	Rw, Rw
2D	2	JMPR	cc_EQ, rel or cc_Z, rel
2E	2	BCLR	bitoff.2
2F	2	BSET	bitoff.2
30	2	SUBC	Rw, Rw
31	2	SUBCB	Rb, Rb
32	4	SUBC	reg, mem
33	4	SUBCB	reg, mem
34	4	SUBC	mem, reg
35	4	SUBCB	mem, reg
36	4	SUBC	reg, #data16
37	4	SUBCB	reg, #data8
38	2	SUBC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
39	2	SUBCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
3A	4	BMOVN	bitaddr, bitaddr
3B	-	-	-
3C	2	ROR	Rw, #data4
3D	2	JMPR	cc_NE, rel or cc_NZ, rel
3E	2	BCLR	bitoff.3
3F	2	BSET	bitoff.3
40	2	CMP	Rw, Rw
41	2	СМРВ	Rb, Rb
42	4	CMP	reg, mem
43	4	СМРВ	reg, mem
44	-	-	-
45	-	-	-
46	4	CMP	reg, #data16
47	4	СМРВ	reg, #data8
48	2	CMP	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
49	2	СМРВ	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
4A	4	BMOV	bitaddr, bitaddr
4B	2	DIV	Rw
4C	2	SHL	Rw, Rw

 Table 4.16
 Instruction set ordered by Hex code (cont'd)



#### Instruction Set Summary

Hex- code	Number of Bytes	Mnemonic	Operand
4D	2	JMPR	cc_V, rel
4E	2	BCLR	bitoff.4
4F	2	BSET	bitoff.4
50	2	XOR	Rw, Rw
51	2	XORB	Rb, Rb
52	4	XOR	reg, mem
53	4	XORB	reg, mem
54	4	XOR	mem, reg
55	4	XORB	mem, reg
56	4	XOR	reg, #data16
57	4	XORB	reg, #data8
58	2	XOR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
59	2	XORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1</sup> )
5A	4	BOR	bitaddr, bitaddr
5B	2	DIVU	Rw
5C	2	SHL	Rw, #data4
5D	2	JMPR	cc_NV, rel
5E	2	BCLR	bitoff.5
5F	2	BSET	bitoff.5
60	2	AND	Rw, Rw
61	2	ANDB	Rb, Rb
62	4	AND	reg, mem
63	4	ANDB	reg, mem
64	4	AND	mem, reg
65	4	ANDB	mem, reg
66	4	AND	reg, #data16
67	4	ANDB	reg, #data8
68	2	AND	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1)</sup>
69	2	ANDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
6A	4	BAND	bitaddr, bitaddr
6B	2	DIVL	Rw
6C	2	SHR	Rw, Rw
6D	2	JMPR	cc_N, rel
6E	2	BCLR	bitoff.6
6F	2	BSET	bitoff.6
70	2	OR	Rw, Rw
71	2	ORB	Rb, Rb
72	4	OR	reg, mem
73	4	ORB	reg, mem

# Table 4.16 Instruction set ordered by Hex code (cont'd)



Hex- code	Number of Bytes	Mnemonic	Operand
74	4	OR	mem, reg
75	4	ORB	mem, reg
76	4	OR	reg, #data16
77	4	ORB	reg, #data8
78	2	OR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 <sup>1</sup> )
79	2	ORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 <sup>1)</sup>
7A	4	BXOR	bitaddr, bitaddr
7B	2	DIVLU	Rw
7C	2	SHR	Rw, #data4
7D	2	JMPR	cc_NN, rel
7E	2	BCLR	bitoff.7
7F	2	BSET	bitoff.7
80	2	CMPI1	Rw, #data4
81	2	NEG	Rw
82	4	CMPI1	Rw, mem
83	-	-	-
84	4	MOV	[Rw], mem
85	-	-	-
86	4	CMPI1	Rw, #data16
87	4	IDLE	
88	2	MOV	[-Rw], Rw
89	2	MOVB	[-Rw], Rb
8A	4	JB	bitaddr, rel
8B	-	-	-
8C	-	-	-
8D	2	JMPR	cc_C, rel or cc_ULT, rel
8E	2	BCLR	bitoff.8
8F	2	BSET	bitoff.8
90	2	CMPI2	Rw, #data4
91	2	CPL	Rw
92	4	CMPI2	Rw, mem
93	-	-	-
94	4	MOV	mem, [Rw]
95	-	-	-
96	4	CMPI2	Rw, #data16
97	4	PWRDN	
98	2	MOV	Rw, [Rw+]
99	2	MOVB	Rb, [Rw+]
9A	4	JNB	bitaddr, rel

 Table 4.16
 Instruction set ordered by Hex code (cont'd)



#### Instruction Set Summary

Hex- code	Number of Bytes	Mnemonic	Operand
9B	2	TRAP	#trap7
9C	2	JMPI	cc, [Rw]
9D	2	JMPR	cc_NC, rel or cc_UGE, rel
9E	2	BCLR	bitoff.9
9F	2	BSET	bitoff.9
A0	2	CMPD1	Rw, #data4
A1	2	NEGB	Rb
A2	4	CMPD1	Rw, mem
A3	-	-	-
A4	4	MOVB	[Rw], mem
A5	4	DISWDT	
A6	4	CMPD1	Rw, #data16
A7	4	SRVWDT	
A8	2	MOV	Rw, [Rw]
A9	2	MOVB	Rb, [Rw]
AA	4	JBC	bitaddr, rel
AB	2	CALLI	cc, [Rw]
AC	2	ASHR	Rw, Rw
AD	2	JMPR	cc_SGT, rel
AE	2	BCLR	bitoff.10
AF	2	BSET	bitoff.10
B0	2	CMPD2	Rw, #data4
B1	2	CPLB	Rb
B2	4	CMPD2	Rw, mem
B3	-	-	-
B4	4	MOVB	mem, [Rw]
B5	4	EINIT	
B6	4	CMPD2	Rw, #data16
B7	4	SRST	
B8	2	MOV	[Rw], Rw
B9	2	MOVB	[Rw], Rb
BA	4	JNBS	bitaddr, rel
BB	2	CALLR	rel
BC	2	ASHR	Rw, #data4
BD	2	JMPR	cc_SLE, rel
BE	2	BCLR	bitoff.11
BF	2	BSET	bitoff.11

# Table 4.16 Instruction set ordered by Hex code (cont'd)



# 4.5 Instruction conventions

This section details the conventions used in the individual instruction descriptions. Each individual instruction description is described in a standard format in separate sections under the following headings:

## 4.5.1 Instruction name

Specifies the mnemonic opcode of the instruction.

#### 4.5.2 Syntax

Specifies the mnemonic opcode and the required formal operands of the instruction. Instructions can have either none, one, two or three operands which are separated from each other by commas:

```
MNEMONIC {op1 {,op2 {,op3 } } }
```

The syntax for the operands of an instruction depend on the selected addressing mode. All of the available addressing modes are summarized at the end of each single instruction description.

# 4.5.3 Operation

**[**]

Describes the instruction operation in symbolic formula or a high level language construct. The following symbols are used to represent data movement, arithmetic or logical operators.

Diadic operations		(opX)		operator (opY)
	$\leftarrow$	(opY)	is	MOVED into (opX)
	+	(opX)	is	ADDED to (opY)
	_	(opY)	is	SUBTRACTED from (opX)
	*	(opX)	is	MULTIPLIED by (opY)
	/	(opX)	is	DIVIDED by (opY)
	^	(opX)	is	logically ANDed with (opY)
	$\vee$	(opX)	is	logically ORed with (opY)
	$\oplus$	(opX)	is	logically EXCLUSIVELY ORed with (opY)
	$\Leftrightarrow$	(opX)	is	COMPARED against (opY)
	mod	(opX)	is	divided MODULO (opY)
Monadic operations		(opY)		operator (opX)
	-	(opX)	is	logically COMPLEMENTED

#### Instruction Set Summary

Missing or existing parentheses signifies that the operand specifies an immediate constant value, an address, or a pointer to an address as follows:

орХ	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX <sub>n</sub> )	Specifies the contents of bit n of opX
((opX))	Specifies the contents of the contents of opX (i.e. opX is used as pointer to the actual operand)
The following	abbreviations are used to describe operands:
СР	Context Pointer register
CSP	Code Segment Pointer register
IP	Instruction Pointer
MD	Multiply/Divide register (32 bits wide, consists of MDH and MDL)
MDL, MDH	Multiply/Divide Low and High registers (each 16 bit wide)
PSW	Program Status Word register
SP	System Stack Pointer register
SYSCON	System Configuration register
С	Carry condition flag in the PSW register
V	Overflow condition flag in the PSW register
SGTDIS	Segmentation Disable bit in the SYSCON register
count	Temporary variable for an intermediate storage of the number of shift or rotate cycles which remain to complete the shift or rotate operation
tmp	Temporary variable for an intermediate result
0, 1, 2,	Constant values due to the data format of the specified operation

# 4.5.4 Data types

Specifies the particular data type according to the instruction. Basically, the following data types are used:

# BIT, BYTE, WORD, DOUBLEWORD

Except for those instructions which extend byte data to word data, all instructions have only one particular data type. Note that the data types mentioned here do not take into account accesses to indirect address pointers or to the system stack which are always performed with word data. Moreover, no data type is specified for System Control Instructions and for those of the branch instructions which do not access any explicitly addressed data.

# 4.5.5 Description

This section describes the operation of the instruction

# 4.5.6 Condition code

The following table summarizes the 16 possible condition codes that can be used within Call and Branch instructions and shows the mnemonic abbreviations, the test executed for a specific condition and the 4-bit condition code number.

Condition Code Mnemonic cc	Test	Description	Condition Code Number c
CC_UC	1 = 1	Unconditional	0h
cc_Z	Z = 1	Zero	2h
cc_NZ	Z = 0	Not zero	3h
cc_V	V = 1	Overflow	4h
cc_NV	V = 0	No overflow	5h
cc_N	N = 1	Negative	6h
cc_NN	N = 1	Not negative	7h
C_C_C	C = 1	Carry	8h
cc_NC	C = 0	No carry	9h
cc_EQ	Z = 1	Equal	2h
cc_NE	Z = 0	Not equal	3h
cc_ULT	C = 1	Unsigned less than	8h
cc_ULE	(Z∨C) = 1	Unsigned less than or equal	Fh
cc_UGE	C = 0	Unsigned greater than or equal	9h
cc_UGT	(Z∨C) = 0	Unsigned greater than	Eh
cc_SLT	(N⊕V) = 1	Signed less than	Ch
cc_SLE	(Z∨(N⊕V)) = 1	Signed less than or equal	Bh
cc_SGE	(N⊕V) = 0	Signed greater than or equal	Dh
cc_SGT	(Z∨(N⊕V)) = 0	Signed greater than	Ah
cc_NET	(Z∨E) = 0	Not equal AND not end of table	1h

Table 4.17Condition codes

# 4.5.7 Condition flags

This section shows the state of the N, C, V, Z and E flags in the PSW register. The resulting state of the flags is represented by the following symbols

Symbol		Description				
*	The flag is set according to the following standard rules					
	N = 1 :	N = 1: MSB of the result is set				
	N = 0 :	MSB of the result is not set				
	C = 1 :	Carry occurred during operation				
	C = 0 :	No Carry occurred during operation				
	V = 1 :	Arithmetic Overflow occurred during operation				
	V = 0 :	No Arithmetic Overflow occurred during operation				
	Z = 1 :	Result equals zero				
	Z = 0 :	Result does not equal zero				
	E = 1 :	Source operand represents the lowest negative number, either 8000h for word				
		data or 80h for byte data.				
	E = 0 :	Source operand does not represent the lowest negative number for the specified				
		data type				
'S'	The flag	g is set according to non-standard rules. Individual instruction pages or the ALU sta-				
	tus flag	s description.				
'_'	The flag	g is not affected by the operation				
'0'	The flag	g is cleared by the operation.				
'NOR'	The flag	g contains the logical NORing of the two specified bit operands.				
'AND'	The flag contains the logical ANDing of the two specified bit operands.					
'OR'	The flag contains the logical ORing of the two specified bit operands.					
'XOR'	The flag contains the logical XORing of the two specified bit operands.					
'B'	The flag	g contains the original value of the specified bit operand.				
' <u>B</u> '	The flag	contains the complemented value of the specified bit operand				
Note:	If the DS	W register was specified as the destination operand of an instruction, the condition				

Table 4.18List of condition flags

Note: If the PSW register was specified as the destination operand of an instruction, the condition flags can not be interpreted as described, because the PSW register is modified according to the data format of the instruction as follows:

For word operations, the PSW register is overwritten with the word result.

For byte operations, the non-addressed byte is cleared and the addressed byte is overwritten. For bit or bit-field operations on the PSW register, only the specified bits are modified.

If the condition flags are not selected as destination bits, they stay unchanged i.e. they maintain the state existing after the previous instruction.

In any case, if the PSW was the destination operand of an instruction, the PSW flags do NOT represent the condition flags of this instruction as usual.

# 4.5.8 Addressing modes

Specifies available combinations of addressing modes. The selected addressing mode combination is generally specified by the opcode of the corresponding instruction. However, there are some arithmetic and logical instructions where the addressing mode combination is not specified by the (identical) opcodes but by particular bits within the operand field.

In the individual instruction description, the addressing mode is described in terms of mnemonic, format and number of bytes.

- **Mnemonic** gives an example of which operands the instruction will accept.
- Format specifies the format of the instruction as used in the assembler listing. Figure 4.1 shows the reference between the instruction format representation of the assembler and the corresponding internal organization of the instruction format (N = nibble = 4 bits). The following symbols are used to describe the instruction formats:

00 <sub>h</sub> through FF <sub>h</sub>	Instruction Opcodes
0, 1	Constant Values
:	Each of the 4 characters immediately following a colon represents a single bit
:ii	2-bit short GPR address (Rwi)
SS	8-bit code segment number (seg).
:##	2-bit immediate constant (#data2)
:.###	3-bit immediate constant (#data3)
С	4-bit condition code specification (cc)
n	4-bit short GPR address (Rwn or Rbn)
m	4-bit short GPR address (Rwm or Rbm)
q	4-bit position of the source bit within the word specified by QQ
Z	4-bit position of the destination bit within the word specified by ZZ
#	4-bit immediate constant (#data4)
QQ	8-bit word address of the source bit (bitoff)
rr	8-bit relative target address word offset (rel)
RR	8-bit word address reg
ZZ	8-bit word address of the destination bit (bitoff)
##	8-bit immediate constant (#data8)
@@	8-bit immediate constant (#mask8)
рр 0:00рр	10-bit page address (#pag10)
MM MM	16-bit address (mem or caddr; low byte, high byte)
## ##	16-bit immediate constant (#data16; low byte, high byte)

 Table 4.19
 Instruction format symbols

• **Number of bytes** Specifies the size of an instruction in bytes. All ST10 instructions are either 2 or 4 bytes.Instructions are classified as either single word or double word instructions.

#### Individual Instruction Descriptions



#### Figure 4.1 Instruction format representation

# 4.6 Notes on ATOMIC and EXTended instructions

ATOMIC, EXTR, EXTP, EXTS, EXTPR, EXTSR instructions disable standard and PEC interrupts and class A traps during a sequence of the following 1...4 instructions. The length of the sequence is determined by an operand (op1 or op2, depending on the instruction). The EXTended instructions also change the addressing mechanism during this sequence (see detailed instruction description).

The ATOMIC and EXTended instructions become active immediately, so no additional NOPs are required. All instructions requiring multiple cycles or hold states to be executed are regarded as one instruction in this sense. Any instruction type can be used with the ATOMIC and EXTended instructions.

**CAUTION:** When a Class B trap interrupts an ATOMIC or EXTended sequence, this sequence is terminated, the interrupt lock is removed and the standard condition is restored, before the trap routine is executed! The remaining instructions of the terminated sequence that are executed after returning from the trap routine, will run under standard conditions!

**CAUTION:** Be careful, when using the ATOMIC and EXTended instructions with other system control or branch instructions.

**CAUTION:** Be careful, when using nested ATOMIC and EXTended instructions. There is ONE counter to control the length of this sort of sequence, i.e. issuing an ATOMIC or EXTended instruction within a sequence will reload the counter with value of the new instruction.

The ATOMIC and EXTended instructions are not available in the ST10X166 devices.

# 5 Individual Instruction Descriptions

The following pages of this section contain a detailed description of each instruction listed in alphabetical order.

ADD	Integer Addition							
Syntax	ADD op1, op2							
Operation	$(op1) \leftarrow (op1)$	op1) + (op2)						
Data Types	WORD							
Description	Performs operand specified	a 2's complement bina specified by op2 and by op1. The sum is then s	ry addition of the the destination stored in op1.	e source operand				
Condition Flags	E *	Z V C N * * * *						
	Е	Set if the value of op possible negative num Used to signal the end of	2 represents the ober. Cleared ot of a table.	e lowest herwise.				
	Z	Set if result equals zero.	Cleared otherwise	e.				
	V	Set if an arithmetic ov result cannot be represe type. Cleared otherwise.	verflow occurred, ented in the specif	i.e. the ied data				
	С	Set if a carry is general cant bit of the specified wise.	ted from the mos data type. Cleare	t signifi- d other-				
	Ν	Set if the most significated cleared otherwise.	ant bit of the resu	lt is set.				
Addressing Modes	Mnemonio	;	Format	Bytes				
	ADD ADD ADD ADD ADD ADD ADD	Rw <sub>n</sub> , Rw <sub>m</sub> Rw <sub>n</sub> , [Rw <sub>i</sub> ] Rw <sub>n</sub> , [Rw <sub>i</sub> +] Rw <sub>n</sub> , #data <sub>3</sub> reg, #data <sub>16</sub> reg, mem mem, reg	00 nm 08 n:10ii 08 n:11ii 08 n:0### 06 RR ## ## 02 RR MM MM 04 RR MM MM	2 2 2 2 4 4 4				

#### ADDB

ADDB	Integer Addition								
Syntax	ADDB op1, op2								
Operation	$(op1) \leftarrow (a)$	op1) + (op2)							
Data Types	BYTE								
Description	Performs a 2's complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The sum is then stored in op1.								
Condition Flags	E *	Z V C N * * * *							
	E	Set if the value of op possible negative num Used to signal the end of	2 represents the ber. Cleared ot f a table.	e lowest herwise.					
	Z	Set if result equals zero.	Cleared otherwise	e.					
	V	Set if an arithmetic ov result cannot be represe type. Cleared otherwise.	verflow occurred, ented in the specif	i.e. the fied data					
	С	Set if a carry is general cant bit of the specified wise.	ted from the mos data type. Cleare	at signifi- ad other-					
	Ν	Set if the most significated cleared otherwise.	ant bit of the resu	lt is set.					
Addressing Modes	Mnemonio	2	Format	Bytes					
	ADDB	Rb <sub>n</sub> , Rb <sub>m</sub>	01 nm	2					
	ADDB	Rb <sub>n</sub> , [Rw <sub>i</sub> ]	09 n:10ii	2					
	ADDB	Rb <sub>n</sub> , [Rw <sub>i</sub> +]	09 n:11ii	2					
	ADDB	Rb <sub>n</sub> , #data <sub>3</sub>	09 n:0###	2					
	ADDB	reg, #data <sub>16</sub>	07 RR ## ##	4					
		reg, mem	03 RR MM MM	4					
	ADDR	mem, reg	UD KK IVIIVI IVIIVI	4					

ADDC	Integer A	dditio	n with	Carry					
Syntax	ADDC	ADDC op1, op2							
Operation	$(op1) \leftarrow ($	op1) +	(op2)	+ (C)					
Data Types	WORD								
Description	Performs a 2's complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.								
Condition Flags	E	7	v	c	NI				
	E *	S	*	*	*				
	E	Set if negat the er	the va ive nu nd of a	lue of oj mber. C table.	o2 rep leared	resents the lowe d otherwise. Use	st possible d to signal		
	Z	Set if Clear	result ed oth	equals : erwise.	zero a	nd previous Z fla	ag was set.		
	V	Set if canno Clear	an ari ot be i ed oth	thmetic represe erwise.	overfl nted i	ow occurred, i.e n the specified	. the result data type.		
	С	Set if bit of	a carr the sp	y is ger ecified o	nerate data ty	d from the most pe. Cleared othe	significant erwise.		
	Ν	Set if Clear	<sup>:</sup> the r ed oth	nost sig erwise.	gnifica	nt bit of the rea	sult is set.		
Addressing Modes	Mnemonio	C			F	Format	Bytes		
-	ADDC	Rw <sub>n</sub> ,	Rw <sub>m</sub>		1	l0 nm	2		
	ADDC	Rw <sub>n</sub> ,	[Rw <sub>i</sub> ]		1	18 n:10ii	2		
	ADDC	Rw <sub>n</sub> ,	[Rw <sub>i</sub> +]		1	18 n:11ii	2		
	ADDC	Rw <sub>n</sub> ,	#data <sub>3</sub>	5	1	l8 n:0###	2		
	ADDC	reg, #	data <sub>16</sub>	5	1	I6 RR ## ##	4		
	ADDC	reg, n	nem		1	12 RR MM MM	4		
	ADDC	mem,	reg		1	14 RR MM MM	4		

#### ADDBC

ADDBC	Integer A	dditic	on with	Carry			
Syntax	ADDBC	op1,	op2				
Operation	$(op1) \leftarrow (op1)$	эр1) н	- (op2) ·	+ (C)			
Data Types	BYTE						
Description	Performs a 2's complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.						
Condition Flags	-	7	V	0	N		
	<b>E</b>	Z S	V *	*	N *	7	
	E	Set if nega the e	f the val tive nui end of a	ue of op mber. C table.	o2 repr leared	esents the lowes otherwise. Used	st possible d to signal
	Z	Set i Clea	f result red othe	equals z erwise.	zero ar	nd previous Z fla	g was set.
	V	Set i cann Clea	f an ari ot be r red othe	thmetic epreser erwise.	overflonted in	w occurred, i.e. the specified	the result data type.
	С	Set i bit of	f a carr the spe	y is ger ecified c	nerated lata typ	from the most be. Cleared othe	significant rwise.
	Ν	Set Clea	if the n red othe	nost sig erwise.	nifican	t bit of the res	sult is set.
Addressing Modes	Mnemonic ADDCB ADDCB ADDCB ADDCB ADDCB	Rb <sub>n</sub> , Rb <sub>n</sub> , Rb <sub>n</sub> , Rb <sub>n</sub> , reg, s	Rb <sub>m</sub> [Rw <sub>i</sub> ] [Rw <sub>i</sub> +] #data <sub>3</sub> #data <sub>16</sub>		F( 1) 1) 1) 1) 1) 1)	ormat 1 nm 9 n:10ii 9 n:11ii 9 n:0### 7 RR ## ## 3 RR MM MM	Bytes 2 2 2 2 2 4 4
	ADDCB	mem	, reg		1	5 RR MM MM	4



AND	Logical AND						
Syntax	AND	op1, op2					
Operation	$(op1) \leftarrow ($	op1) ^ (op2)					
Data Types	WORD						
Description	Performs by op2 an is then sto	Performs a bitwise logical AND of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.					
Condition Flags	-	7 1 0 1					
	E *	* 0 0 *					
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.					
	Z	Set if result equals zero.	Cleared otherwise	<b>)</b> .			
	V	Always cleared.					
	С	Always cleared.					
	Ν	Set if the most signific Cleared otherwise.	ant bit of the res	ult is set.			
Addressing Modes	Mnemonio	2	Format	Bytes			
	AND	Rw <sub>n</sub> , Rw <sub>m</sub>	60 nm	2			
	AND	Rw <sub>n</sub> , [Rw <sub>i</sub> ]	68 n:10ii	2			
	AND	68 n:11ii	2				
	AND	Rw <sub>n</sub> , #data <sub>3</sub>	68 n:0###	2			
	AND	reg, #data <sub>16</sub>	66 RR ## ##	4			
	AND AND	reg, mem mem, reg	62 RR MM MM 64 RR MM MM	4 4			

#### ANDB

ANDB	Logical	AND							
Syntax	ANDB	ANDB op1, op2							
Operation	(op1) ←	(op1) ^ (op2)							
Data Types	BYTE								
Description	Performs by op2 a is then st	a bitwise log nd the destina tored in op1.	ical AN	D of the erand s	e source operan pecified by op1	nd specified . The result			
Condition Flags	Е	z v	С	N					
	*	* 0	0	*					
	Е	Set if the va negative nu the end of a	lue of o mber. ( table.	p2 repr Cleared	esents the lowe otherwise. Use	est possible ed to signal			
	Z	Set if result	equals	zero. C	leared otherwis	se.			
	V	Always clea	red.						
	С	Always clea	red.						
	Ν	Set if the i Cleared oth	most si erwise.	gnificar	nt bit of the re	sult is set.			
Addressing Modes	Mnemon	ic		F	ormat	Bytes			
	ANDB	Rb <sub>n</sub> , Rb <sub>m</sub>		6	1 nm	2			
	ANDB	Rb <sub>n</sub> , [Rw <sub>i</sub> ]		6	9 n:10ii	2			
	ANDB	Rb <sub>n</sub> , [Rw <sub>i</sub> +]		6	9 n:11ii	2			
	ANDB	Rb <sub>n</sub> , #data <sub>3</sub>	5	6	9 n:0###	2			
	ANDB	reg, #data <sub>10</sub>	6	6	7 RR ## ##	4			
	ANDB	reg, mem		6	3 RR MM MM	4			
	ANDB	mem, reg		6	5 RR MM MM	4			

mem, reg

ASHR	Arithmeti	c Shift I	Right	:			
Syntax	ASHR	op1, op	2				
Operation	$\begin{array}{l} (\text{count}) \leftarrow \\ (V) \leftarrow 0 \\ (C) \leftarrow 0 \\ \text{DO WHILL} \\ (V) \leftarrow (C) \\ (C) \leftarrow (\text{op} \\ (\text{op} 1_n) \leftarrow \\ (\text{count}) \leftarrow \\ \text{END WHILL} \end{array}$	$(op1) \land$ E (count $V \lor (V)$ $O1_0)$ $(op1_{n+1})$ LE	(op2 t) ≠ 0 ) [n= ) - 1	) 014]			
Data Types	WORD						
Description	Arithmetically shifts the destination word operand op1 right by as many times as specified in the source operand op2. To preserve the sign of the original operand op1, the most signifi- cant bits of the result are filled with zeros if the original MSB was a 0 or with ones if the original MSB was a 1. The Overflow flag is used as a Rounding flag. The LSB is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.						
Condition Flags	_	-	.,	•			
	E	<b>Z</b>	v S	S	N *		
	E Z V	Always cleared. Set if result equals zero. Cleared otherwise. Set if in any cycle of the shift operation a 1 is shifted out of the carry flag. Cleared for a shift count of zero					
	C The carry flag is set according to the last LSB shift out of op1. Cleared for a shift count of zero.						LSB shifted ro.
	N Set if the most significant bit of the result is Cleared otherwise.						esult is set.
Addressing Modes	Mnemonic	) ,			F	Format	Bytes
	ASHR	кw <sub>n</sub> , к Rw <sub>n</sub> , #	w <sub>m</sub> data <sub>4</sub>		Ĩ	3C #n	2 2

#### ATOMIC

ATOMIC	Begin ATOMIC Sequence							
Syntax	ATOMIC	op1						
Operation	(count) ( Disable DO WHI Next Ins (count) END WH (count) = Enable i	count) ← (op1) $[1 \le op1 \le 4]$ visable interrupts and Class A traps O WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE) Next Instruction count) ← (count) - 1 ND WHILE count) = 0 inable interrupts and traps						
Description	Causes standard and PEC interrupts and class A hardware traps to be disabled for a specified number of instructions. The ATOMIC instruction becomes immediately active so that no additional NOPs are required. Depending on the value of op1, the period of validity of the ATOMIC sequence extends over the sequence of the next 1 to 4 instructions being executed after the ATOMIC instruction. All instructions requiring multiple cycles or hold states to be executed are regarded as one instruction in this sense. Any instruction type can be used with the ATOMIC instruction							
Note	The ATC 4.6 "Not The ATC	DMIC i es on A DMIC ir	nstructio ATOMIC Istructio	on mus and EX n is not	t be us (Tende availat	ed carefully d instructions ble for ST10X	(see Section "). 166 devices.	
Condition Flags								
-	E	Z	V	С	Ν	7		
	-	-	-	-	-			
	E	Not	affected					
	Z	Not	affected					
	V	Not	affected					
	С	Not	affected					
	Ν	Not	affected					
Addressing Modes	Mnemor ATOMIC	nic #dat	a <sub>2</sub>		F D	ormat 1 :00##-0	Bytes 2	



BAND	Bit Logical AND						
Syntax	BAND op1, op2						
Operation	$(op1) \leftarrow (op1) \land (op2)$						
Data Types	BIT						
Description	Performs a single bit logical AND of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.						
Condition Flags							
	E Always cleared.						
	Z Contains the logical NOR of the two specified bits.						
	V Contains the logical OR of the two specified bits.						
	C Contains the logical AND of the two specified bits.						
	N Contains the logical XOR of the two specified bits.						
Addressing Modes	MnemonicFormatBytesBAND bitaddr <sub>Z.z</sub> , bitaddr <sub>Q.q</sub> 6A QQ ZZ qz4						

#### BCLR

BCLR	Bit Clear						
Syntax	BCLR	op1					
Operation	(op1) ← 0						
Data Types	BIT						
Description	CLears th used for p	e bit eriphe	specifie eral and	ed by a system	p1. T contr	his instruction ol.	is primarily
Condition Flags	-	7	V	0	N		
	Е 0	Z B	<b>v</b>	0	B	7	
	E	Alwa	ys clear	ed.			
	Z	Contact the s	ains the pecified	logical I bit.	negat	ion of the previo	ous state of
	V	Alwa	ys clear	ed.			
	С	Alwa	ys clear	ed.			
	Ν	Cont	ains the	e previou	us stat	te of the specifie	ed bit.
Addressing Modes	Mnemonic BCLR	; bitad	dr <sub>Q.</sub>		F	Format JE QQ	Bytes 2

BCMP	Bit to Bit Co	ompare			
Syntax	BCMP op	o1, op2			
Operation	(op1) ⇔ (op2	2)			
Data Types	BIT				
Description	Performs a s operand op1 result is writt updated.	single bit co 1 to the so ten by this	omparison ource bit s instruction	of the source bit s pecified by operar . Only the conditior	specified by nd op2. No n codes are
Condition Flags	EZ	V R OR	C AND X	N DR	
	E A	lways clear	ed.		
	Z C	ontains the	logical NC	OR of the two spec	ified bits.
	V C	ontains the	logical OF	R of the two specifi	ed bits.
	C C	ontains the	logical AN	ID of the two spec	ified bits.
	N C	ontains the	logical XC	OR of the two spec	ified bits.
Addressing Modes	Mnemonic BCMP bi	taddr <sub>Z.z</sub> , bi	taddr <sub>Q.q</sub>	Format 2A QQ ZZ qz	Bytes 4

#### BFLDH

BFLDH	Bit Field High Byte							
Syntax	BFLDH	BFLDH op1, op2, op3						
Operation	$(tmp) \leftarrow (d)$ (high byte (op1) $\leftarrow (t)$	op1) (tmp] mp)	)) ← ((h	igh byte	e (tmp)	∧ ¬op2) ∨ op3)		
Data Types	WORD							
Description	Replaces those bits in the high byte of the destination word operand op1 which are selected by an '1' in the AND mask op2 with the bits at the corresponding positions in the OR mask specified by op3.							
Note	Bits which are masked off by a '0' in the AND mask op2 may be unintentionally altered if the corresponding bit in the OR mask op3 contains a '1'.							
Condition Flags	_	_		•				
	<b>E</b>	<b>۲</b> *	<b>V</b> 0	0	N *			
	E	Alwa	iys clear	ed.				
	Z	Set i	f the wo	rd resul	t equa	ls zero. Cleared	otherwise.	
	V	Alwa	iys clear	ed.				
	С	Alwa	iys clear	ed.				
	Ν	Set if the most significant bit of the word result is set. Cleared otherwise.						
Addressing Modes	Mnemonic BFLDH bit	; toff <sub>Q</sub> ,	#mask <sub>8</sub>	, #data	8	Format 1A QQ ## @@	Bytes 4	

BFLDL	Bit Field I	eld Low Byte							
Syntax	BFLDL	op1,	op2, op	3					
Operation	$(tmp) \leftarrow (contraction (low byte (low byte (low (low 1)))))$	op1) (tmp)) mp)	$\leftarrow ((Iov$	v byte (	tmp) ∧	–op2) ∨ op3)			
Data Types	WORD								
Description	Replaces those bits in the low byte of the destination word operand op1 which are selected by an '1' in the AND mask op2 with the bits at the corresponding positions in the OR mask specified by op3.								
Note	Bits which are masked off by a '0' in the AND mask op2 may be unintentionally altered if the corresponding bit in the OR mask op3 contains a '1'.								
Condition Flags									
	<b>E</b>	Z *	<b>V</b> 0	<b>C</b>	<b>N</b> *	]			
	E	Alwa	ys cleai	ed.					
	Z	Set if	the wo	rd resul	t equa	s zero. Cleared otherwise.			
	V	Alwa	ys cleai	ed.					
	С	Alwa	ys cleai	ed.					
	Ν	Set if Clea	f the mo red othe	ost signi erwise.	ficant k	bit of the word result is set.			
Addressing Modes	Mnemonic BFLDL bi	: toff <sub>Q</sub> ,	#mask <sub>8</sub>	<sub>3</sub> , #data	8	Format Bytes 0A QQ @@## 4			

#### BMOV

BMOV	Bit to Bit Move								
Syntax	BMOV	op1, op2							
Operation	(op1) ← (op2)								
Data Types	BIT								
Description	Moves a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.								
Condition Flags									
	Conditi		0	B	0	0	B		
	E	Always clea	ared.						
	Z	Contains the logical negation of the previous state of the source bit.							
	V	Always clea	ared.						
	С	Always clea	ared.						
	Ν	N Contains the previous state of the source bit.							
Addressing Modes	Mnemonic BMOV bita	; addr <sub>Z.z</sub> , bitac	ddr <sub>Q.q</sub>	ormat A QQ Z	Z qz	Bytes 4			

BMOVN	Bit to Bit Move and Negate								
Syntax	BMOVN	BMOVN op1, op2							
Operation	$(op1) \leftarrow -$	(op2)							
Data Types	BIT								
Description	Moves the complement of a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.								
Condition Flags	<b>E</b>	<b>z V</b> B C	)	<b>C</b> 0	N B				
	E	Always o	cleared	d.					
	Z	Contains the sour	s the lo ce bit.	ogical	negat	ion of the previ	ous state of		
	V	Always o	cleared	d.					
	С	Always cleared.							
	Ν	Contains	s the p	orevio	us stat	e of the source	e bit.		
Addressing Modes	Mnemonio BMOVN b	; pitaddr <sub>Z.z</sub>	, bitad	dr <sub>Q.q</sub>	F 3	ormat A QQ ZZ qz	Bytes 4		

## BOR

BOR	Bit Logica	al OR						
Syntax	BOR	op1, c	op2					
Operation	$(op1) \leftarrow (a)$	$(op1) \leftarrow (op1) \lor (op2)$						
Data Types	BIT							
Description	Performs a single bit logical OR of the source bit specified by operand op2 with the destination bit specified by operand op1. The ORed result is then stored in op1.							
Condition Flags	<b>E</b> 0 N	<b>z</b> IOR	<b>V</b> OR	<b>C</b> AND	N XOR	]		
	E	Alway	vs clear	ed.				
	Z	Conta	ains the	logical	NOR o	of the two spec	ified bits.	
	V	Conta	ains the	logical	OR of	the two specifi	ed bits.	
	С	Conta	ains the	logical	AND c	of the two spec	ified bits.	
	Ν	Conta	ains the	logical	XOR o	of the two spec	ified bits.	
Addressing Modes	Mnemonic BOR bita	; ddr <sub>Z.z</sub> ,	bitadd	r <sub>Q.q</sub>	Fo 5/	ormat A QQ ZZ qz	Bytes 4	



BSET	Bit Set								
Syntax	BSET	op1							
Operation	(op1) ← 1								
Data Types	BIT								
Description	Sets the b for periphe	oit spe eral ar	cified b nd syste	em cont	This ins rol.	struction is pri	marily used		
Condition Flags	_	-	.,	•					
	E	Z	V	С	N				
	0	В	0	0	В				
	E	Alwa	ys clear	ed.					
	Z	Conta the s	ains the pecified	e logical I bit.	negatio	on of the previ	ous state of		
	V	Alway	ys clear	ed.					
	С	Alwa	ys clear	ed.					
	Ν	N Contains the previous state of the specified bit.							
Addressing Modes	Mnemonic BSET	; bitade	dr <sub>Q.q</sub>		Fc qF	ormat QQ	Bytes 2		

### **BXOR**

BXOR	Bit Logical XOR							
Syntax	BXOR op1, op2							
Operation	$(op1) \leftarrow (op1) \oplus (op2)$							
Data Types	BIT							
Description	Performs a single bit logical EXCLUSIVE OR of the source bit specified by operand op2 with the destination bit specified by operand op1. The XORed result is then stored in op1.							
Condition Flags	EZVCN0NORORANDXOR							
	E Always cleared.							
	Z Contains the logical NOR of the two specified bits.							
	V Contains the logical OR of the two specified bits.							
	C Contains the logical AND of the two specified bits.							
	N Contains the logical XOR of the two specified bits.							
Addressing Modes	MnemonicFormatBytesBXOR bitaddr <sub>Z.z</sub> , bitaddr <sub>Q.q</sub> 7A QQ ZZ qz4							



CALLA	Call Subr	outin	e Absol	ute					
Syntax	CALLA	op1,	op2						
Operation	$\begin{array}{l} IF (op1) T \\ (SP) \leftarrow (S \\ ((SP)) \leftarrow \\ (IP) \leftarrow op \\ ELSE \\ next instru \\ END IF \end{array}$	HEN P) - 2 (IP) 2 uction	2						
Description	If the condition specified by op1 is met, a branch to the absolute memory location specified by the second operand op2 is taken. The value of the instruction pointer, IP, is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.								
<b>Condition Codes</b>	See cond	ition c	ode Tab	le 4.17	on p	age 45.			
Condition Flags	Е	z	v	с	N				
	-	-	-	-	-				
	E	Not	affected.						
	Z	Not	affected.						
	V	Not affected.							
	С	Not	affected.						
	Ν	Not	affected.						
Addressing Modes	Mnemonio CALLA	с сс, с	addr		Format CA c0 MM MM	Bytes VI 4			

# CALLI

CALLI	Call S	Subro	outin	e Indire	ect				
Syntax	CALL	I (	op1,	op2					
Operation	IF (op (SP) ∢ ((SP)) (IP) ← ELSE next ii END	1) TH ← (SF ) ← (I - (op2 nstruc	HEN P) - 2 P) 2) ction	2					
Description	If the condition specified by op1 is met, a branch to the location specified indirectly by the second operand op2 is taken. The value of the instruction pointer, IP, is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.								
Condition Codes	See c	onditi	ion c	ode Tal	ole 4.	17 on j	bag	e 45.	
Condition Flags	_		_		-				
	E -		Z -	V -	C -	N			
	E 7		NOT 8						
	Z Not affected.								
			NOL à						
	IN .		INOL &	anecteo			-	1	
Addressing Modes	Mnemonic Format By CALLI cc, [Rw <sub>n</sub> ] AE						Bytes AB cn2		



CALLR	Call Subr	outin	e Relat	ive					
Syntax	CALLR	op1							
Operation	$(SP) \leftarrow (SP) - 2$ $((SP)) \leftarrow (IP)$ $(IP) \leftarrow (IP) + sign_extend (op1)$								
Description	A branch is taken to the location specified by the instruction pointer, IP, plus the relative displacement, op1. The displace- ment is a two's complement number which is sign extended and counts the relative distance in words. The value of the instruction pointer (IP) is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack repre- sents the return address of the calling routine. The value of the IP used in the target address calculation is the address of the instruction following the CALLR instruction.								
Condition Codes	See condi	tion c	ode Tab	ole 4.17	on pag	ge 45.			
Condition Flags	E	Z	v	С	N	1			
Addressing Modes	E     Not affected.       Z     Not affected.       V     Not affected.       C     Not affected.       N     Not affected.       Mnemonic     Format       Bytes								
	UALLK	iei			В	DII	Z		

69/185

#### CALLS

CALLS	Call Inter-Segment Subroutine								
Syntax	CALLS	op1,	op2						
Operation	$(SP) \leftarrow (S) \\ ((SP)) \leftarrow (S) \\ (SP) \leftarrow (S) \\ ((SP)) \leftarrow (S) \\ (CSP) \leftarrow (S) \\ (IP) \leftarrow Op $	SP) - 2 (CSP) SP) - 2 (IP) op1 2	2						
Description	A branch is taken to the absolute location specified by op2 within the segment specified by op1. The value of the instruction pointer (IP) is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address to the calling routine. The previous value of the CSP is also placed on the system stack to insure correct return to the calling segment.								
Condition Codes	See cond	ition c	ode Tab	le 4.17	on p	age 45.			
Condition Flags	E -	Z -	V -	C -	N -				
	7	Not a	affected.						
	V Not affected.								
	С	Not a	affected.						
	Ν	Not a	affected.						
Addressing Modes	Mnemonio CALLS	c seg,	caddr			Format DA ss MM MM	Bytes 4		



$\mathbf{\Gamma}$	RЛ	D
5	Ινι	Г

**Integer Compare** 

Syntax CMP op1, op2

 $\textbf{Operation} \qquad (op1) \Leftrightarrow (op2)$ 

Data Types WORD

Description

The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

#### **Condition Flags**

Condition 1 lags	Е	Z	V	С	Ν				
	*	*	*	S	*				
	Е	Set if the value of op2 represents the lowest poss negative number. Cleared otherwise. Used to sig the end of a table.							
	Z	Set if result equals zero. Cleared otherwise.							
	V	Set if an arithmetic underflow occurred, i.e. the resul cannot be represented in the specified data type Cleared otherwise.							
	С	Set i	f a borr	ow is ge	enerate	d. Cleared othe	erwise.		
	Ν	Set Clea	if the r red oth	nost si erwise.	gnifican	t bit of the re	sult is set.		
Addressing Modes	Mnemor	nic			F	ormat	Bytes		
-	CMP	Rw <sub>n</sub>	, Rw <sub>m</sub>		4	0 nm	2		
	CMP	Rw <sub>n</sub>	, [Rw <sub>i</sub> ]		4	8 n:10ii	2		
	CMP	Rw <sub>n</sub>	, [Rw <sub>i</sub> +]		4	8 n:11ii	2		
	CMP	Rw <sub>n</sub>	, #data <sub>3</sub>	3	4	8 n:0###	2		
	CMP	reg,	#data <sub>16</sub>	6	4	6 RR ## ##	4		
	CMP	reg,	mem		4	2 RR MM MM	4		

#### CMPB

СМРВ	Integer Compare						
Syntax	CMPB op1, op2						
Operation	$(op1) \Leftrightarrow (op2)$						
Data Types	BYTE						
Description	The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged						
Condition Flags	_	7	M	0			
	E *	*	¥	S	N *		
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.					
	Z	Set if result equals zero. Cleared otherwise.					
	V	Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.					
	С	Set if a borrow is generated. Cleared otherwise.					
	Ν	Set if the most significant bit of the result is set. Cleared otherwise.					
Addressing Modes	Mnemonic CMPB CMPB CMPB CMPB	Rb <sub>n</sub> , Rb <sub>n</sub> , Rb <sub>n</sub> , Rb <sub>n</sub> ,	Rb <sub>m</sub> [Rw <sub>i</sub> ] [Rw <sub>i</sub> +] #data <sub>3</sub>			Format 41 nm 49 n:10ii 49 n:11ii 49 n:0### 47 RR ## ##	Bytes 2 2 2 2 2 4
	CMPB	reg, mem				43 RR MM MM	4
## CMPD1

Integer Compare and Decrement by 1

	_				-	
Syntax	CMPD1	op1,	op2			
Operation	(op1) ⇔ (op1) ←	> (op2) - (op1) -	- 1			
Data Types	WORD					
Description	This ins bility of compar- perform op1. Op subtract one. Us in conju languag	truction f loops ed to ing a 2 berand o tion has ing the inction je FOR	n is used the so c's compop1 may comple set flag with this loops o	d to enh source plement y specif eted, the s, a bra s instru f any ra	nance th operance t binary y ONLY e operan nch ins ction to nge.	he performance and flexi- nd specified by op1 is d specified by op2 by y subtraction of op2 from GPR registers. Once the nd op1 is decremented by truction can then be used form common high level
Condition Flags	-	-	V	•	N	
	E	2	V	C	N	1
	*	*	*	S	*	

Е	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
Z	Set if result equals zero. Cleared otherwise.
V	Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.
С	Set if a borrow is generated. Cleared otherwise.
Ν	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes	Mnemonie	c	Format	Bytes
	CMPD1	Rw <sub>n</sub> , #data <sub>4</sub>	A0 #n	2
	CMPD1	Rw <sub>n</sub> , #data <sub>16</sub>	A6 Fn ## ##	4
	CMPD1	Rw <sub>n</sub> , mem	A2 Fn MM MM	4
	CIMPDI	кw <sub>n</sub> , mem		4

#### CMPD2

CMPD2	Integer C	ompa	are and	Decrer	nent k	oy 2	
Syntax	CMPD2	op1,	op2				
Operation	$(op1) \Leftrightarrow (op1) \Leftrightarrow (op1) \leftarrow (op1$	op2) op1) -	· 2				
Data Types	WORD						
Description	This instruction is used to enhance the performance and flexi- bility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is decremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.						
Condition Flags	_	_					
	<b>E</b>	Ζ *	<b>V</b> *	C S	N *		
	E	Set i nega the e	f the val ative nui end of a	ue of o mber. C table.	p2 rep learec	resents the lowe I otherwise. Use	est possible ed to signal
	Z	Set i	f result	equals	zero. C	Cleared otherwis	e.
	V	Set if an arithmetic underflow occurred, i.e. the resul cannot be represented in the specified data type Cleared otherwise.					
	С	Set i	f a borro	ow is ge	enerate	ed. Cleared othe	rwise.
	Ν	Set Clea	if the n red othe	nost się erwise.	gnifica	nt bit of the rea	sult is set.
Addressing Modes	Mnemonic CMPD2 CMPD2 CMPD2	Rw <sub>n</sub> Rw <sub>n</sub> Rw <sub>n</sub>	, #data <sub>4</sub> , #data <sub>1</sub> , mem	6	F E E	Format 30 #n 36 Fn ## ## 32 Fn MM MM	Bytes 2 4 4



### CMPI1

Description

Integer Compare and Increment by 1

Syntax	CMPI1	op1, op2
Operation	(op1) ⇔	(op2)

 $(op1) \Leftrightarrow (op2)$  $(op1) \leftarrow (op1) + 1$ 

CMPI1

Rw<sub>n</sub>, mem

Data Types WORD

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by one. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

#### **Condition Flags**

J	Е	Ζ	V	С	Ν				
	*	*	*	S	*				
	E	Set i nega the e	if the val ative nur end of a	ue of or nber. C table.	o2 repi leared	resents the lov otherwise. U	west possible sed to signa	) 	
	Z	Set	if result	equals :	zero. C	leared otherv	vise.		
	V	Set i canr Clea	if an arit not be r ared othe	hmetic ( epresei erwise.	underfl nted ir	ow occurred, the specifie	i.e. the resuled data type	t	
	С	Set	if a borro	ow is ge	enerate	ed. Cleared ot	herwise.		
	Ν	Set if the most significant bit of the result is set. Cleared otherwise.							
Addressing Modes	Mnemon	nic			F	ormat	Bytes		
-	CMPI1	Rw <sub>n</sub>	, #data <sub>4</sub>		8	0 #n	2		
	CMPI1	Rwn	, #data₁	6	8	6 Fn ## ##	4		

82 Fn MM MM

57

#### CMPI2

CMPI2	Integer C	ompa	are and	Increm	nent b	y 2				
Syntax	CMPI2 op1, op2									
Operation	$(op1) \Leftrightarrow (op1) \leftarrow (op1)$	op2) op1) -	+ 2							
Data Types	WORD									
Description	This instruction is used to enhance the performance and flexi- bility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.									
Condition Flags				_						
	E *	Z *	<b>V</b>	C S	N *					
	E	Set i nega the e	f the val ative nui end of a	lue of oj mber. C table.	p2 rep leared	resents the lowe I otherwise. Use	est possible ed to signal			
	Z	Set i	f result	equals	zero. (	Cleared otherwis	se.			
	V	Set i cann Clea	f an arit not be r red othe	hmetic represe erwise.	underf nted i	low occurred, i.e n the specified	e. the result data type.			
	С	Set i	f a borro	ow is ge	enerate	ed. Cleared othe	erwise.			
	Ν	Set Clea	if the n red othe	nost sią erwise.	gnifica	nt bit of the re	sult is set.			
Addressing Modes Mnemon CMPI2 CMPI2			, #data <sub>4</sub> , #data <sub>1</sub> mem	6	F 9 9	Format 00 #n 06 Fn ## ## 02 Fn MM MM	Bytes 2 4 4			
	CMPI2 Rw <sub>n</sub> , mem 92 Fn MM MM 4									



CPL	Integer O	ne's Co	ompler	nent				
Syntax	CPL	op1						
Operation	$(op1) \leftarrow -$	₁(op1)						
Data Types	WORD							
Description	Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.							
Condition Flags	E *	<b>Z</b> *	<b>V</b>	<b>C</b>	N *			
	E	Set if t negati the en	he value ve numl d of a ta	e of op oer. Cle able.	1 repre eared	esents the l otherwise.	owest possible Used to signal	
	Z	Set if r	esult ec	uals ze	ero. Cl	eared othe	rwise.	
	V	Always	s cleare	d.				
	С	Always	s cleare	d.				
	Ν	Set if Cleare	the mo	st sigr wise.	nificant	t bit of the	e result is set.	
Addressing Modes	Mnemonio CPL	Rw <sub>n</sub>			Fc 91	ormat n0	Bytes 2	

#### CPLB

CPLB	Integer O	ne's Co	mplem	ent				
Syntax	CPL	op1						
Operation	(op1) $\leftarrow$ –	(op1)						
Data Types	BYTE							
Description	Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.							
Condition Flags	E *	Z *	<b>V</b> 0	<b>C N</b>	N *			
	E	Set if the negative the end	ne value /e numb d of a ta	e of op1 r ber. Clear ble.	epresents the lowe red otherwise. Use	st possible d to signal		
	Z	Set if r	esult eq	uals zero	o. Cleared otherwis	e.		
	V	Always	cleared	d.				
	С	Always	cleared	d.				
	Ν	Set if Cleare	the mo d other\	st signifi vise.	cant bit of the res	sult is set.		
Addressing Modes	Mnemonic CPLB	: Rb <sub>n</sub>			Format B1 n0	Bytes 2		



### DISWDT

**Syntax** 

Operation

**Disable Watchdog Timer** 

DISWDT

Disable the watchdog timer

Description This instruction disables the watchdog timer. The watchdog timer is enabled by a reset. The DISWDT instruction allows the watchdog timer to be disabled for applications which do not require a watchdog function. Following a reset, this instruction can be executed at any time until either a Service Watchdog Timer instruction (SRVWDT) or an End of Initialization instruction (EINIT) are executed. Once one of these instructions has been executed, the DISWDT instruction will have no effect. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

#### **Condition Flags**

annen i age	Е	Z	V	С	Ν		
	-	-	-	-	-	]	
	E	Not	affected	l.			
	Z	Not	affected	Ι.			
	V	Not	affected	Ι.			
	С	Not	affected	Ι.			
	Ν	Not	affected	Ι.			
ressing Modes	Mnem DISWI	onic DT			Fo	ormat 5 5A A5 A5	Bytes 4

Addr

79/185

# DIV

DIV	16-by-16	6 Signed Division						
Syntax	DIV	op1						
Operation	$(MDL) \leftarrow (MDH) \leftarrow$	(MDL) (MDL)	/ (op1) mod (d	op1)				
Data Types	WORD							
Description	Performs a signed 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).							
Condition Flags	<b>E</b>	<b>Z</b>	V S	<b>C</b> 0	<b>N</b> *			
	E	Alway	/s clear	ed.				
	Z	Set if	result e	equals :	zero. Cl	eared otherw	wise.	
	V	Set if an arithmetic overflow occurred, i.e. the result cannot be represented in a word data type, or if the divisor (op1) was zero. Cleared otherwise.						
	С	Alway	/s clear	ed.				
	Ν	Set if Clear	the m ed othe	lost sig rwise.	gnificant	t bit of the	result is set.	
Addressing Modes	Mnemonic	;			Fc	ormat	Bytes	
	DIV	Rw <sub>n</sub>	2					



#### 32-by-16 Signed Division

Syntax	DIVL op1
Operation	$(MDL) \gets (MD) \ / \ (op1)$
	$(MDH) \leftarrow (MD) \mod (op1)$

#### Data Types WORD, DOUBLEWORD

Description

Performs an extended signed 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

### **Condition Flags**

Е	Z	V	С	Ν								
0	*	S	0	*	]							
E	Alwa	Always cleared.										
Z	Set i	Set if result equals zero. Cleared otherwise.										
V	Set i cann divis	Set if an arithmetic overflow occurred, i.e. the result cannot be represented in a word data type, or if the divisor (op1) was zero. Cleared otherwise.										
С	Alwa	iys clea	red.									
N	Set Clea	Set if the most significant bit of the result is set. Cleared otherwise.										
Mnemor DIVL	nic Rw <sub>n</sub>			Fc 61	ormat 3 nn	Bytes 2						

#### **Addressing Modes**

DIVLU	32-by-16 Unsigned Division												
Syntax	DIVLU	DIVLU op1											
Operation	$(MDL) \leftarrow$ $(MDH) \leftarrow$	(MD) / (op1) (MD) mod (	) op1)										
Data Types	WORD, D	OUBLEWO	RD										
Description	Performs two word operand o order word stored in t	Performs an extended unsigned 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The unsigned quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).											
	Condit	ion Flags	Е	z	v	С	Ν						
			0	*	S	0	*						
	Е	Always clea	ared.										
	Z	Set if result	t equals	zero. C	Cleared	otherwi	se.						
	V	Set if an arithmetic overflow occurred, i.e. the result cannot be represented in a word data type, or if the divisor (op1) was zero. Cleared otherwise.											
	С	Always cleared.											
	Ν	Set if the Cleared ot	most sig nerwise.	gnifica	nt bit of	f the re	esult is set.						
Addressing Modes	Mnemoni DIVLU	c Rw <sub>n</sub>		F 7	<sup>-</sup> ormat 'B nn		Bytes 2						

DIVU	16-by-	16	Unsi	Unsigned Division								
Syntax	DIVU		op1									
Operation	(MDL) (MDH)	$(MDL) \leftarrow (MDL) / (op1)$ $(MDH) \leftarrow (MDL) \mod (op1)$										
Data Types	WORD	WORD										
Description	Performs an unsigned 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).											
Condition Flags	F		7	v	C	N						
	0		*	S	0	*	]					
	E		Alwa	ays clea	red.							
	Z		Set i	if result	equals	zero. Cl	eared othe	rwise.				
	V		Set canr divis	if an ari not be r sor (op1	ithmetic epreser ) was ze	overflo nted in a ero. Cle	w occurred a word data ared otherw	, i.e. the result type, or if the <i>v</i> ise.				
	С		Alwa	ays clea	red.							
	Ν		Set if the most significant bit of the result is set Cleared otherwise.									
Addressing Modes	Mnem DIVU	onio	c Rw <sub>n</sub>			Fo 51	ormat 3 nn	Bytes 2				

#### EINIT

EINIT	End of Ir	itializ	ation								
Syntax	EINIT										
Operation	End of In	End of Initialization									
Description	This instruction is used to signal the end of the initialization portion of a program. After a reset, the reset output pin RSTOUT is pulled low. It remains low until the EINIT instruction has been executed at which time it goes high. This enables the program to signal the external circuitry that it has successfully initialized the microcontroller. After the EINIT instruction has been executed, execution of the Disable Watchdog Timer instruction (DISWDT) has no effect. To insure that this instruc- tion is not accidentally executed, it is implemented as a protected instruction.										
Condition Flags	_	-		•							
	E	-	V -	-	N						
	E Z V C	Not affected. Not affected. Not affected. Not affected.									
Addressing Medes	N	Not	affected	•			1	Dutes			
Addressing Modes	EINIT	С				⊦orn B5 4	nat A B5 B5	Bytes 4			

EXTR	Begin E	EXTended Register Sequence											
Syntax	EXTR	op1											
Operation	(count) Disable SFR_rar DO WHI Next Ins (count) END WH (count) = SFR_rar Enable i	) $\leftarrow$ (op1) [1 $\leq$ op1 $\leq$ 4] e interrupts and Class A traps range = Extended HILE ((count) $\neq$ 0 AND Class_B_trap_condition $\neq$ TRUE) instruction t) $\leftarrow$ (count) - 1 VHILE ) = 0 range = Standard e interrupts and traps											
Description	Causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The value of op1 defines the length of the effected instruction sequence.												
Note	The EX "Notes o The EXT	「R inst n ATOI ⁻R insti	ruction MIC and ruction i	must be I EXTer s not av	e used nded in /ailable	carefully (see structions"). for ST10X166	Section 4.6 devices.						
Condition Flags													
	E	Z	V	С	N								
	-	-	-	-	-								
	E	Not	affected	•									
	Z	Not	affected	•									
	V	Not affected.											
	С	Not	affected										
A 1 1	N	Not :	attected		_		5.4						
Addressing Modes	Mnemor EXTR	ic Format Bytes #data <sub>2</sub> D1 :10##-0 2											

EXTP	Begin EX	Tende	d Page	Seq	uence	•					
Syntax	EXTP	op1, c	p2								
Operation	(count) ← Disable in Data_Pag DO WHILI Next Instr (count) ← END WHI (count) = 0 Data_Pag Enable int	$\begin{array}{l} \leftarrow (\text{op2}) \ [1 \leq \text{op2} \leq 4] \\ \text{interrupts and Class A traps} \\ \text{age} = (\text{op1}) \\ \text{LE ((count) } \neq 0 \ \text{AND Class}\_B\_\text{trap\_condition} \neq \text{TRUE}) \\ \text{struction} \\ \leftarrow (\text{count}) - 1 \\ \text{HLE} \\ = 0 \\ \text{age} = (\text{DPPx}) \\ \text{nterrupts and traps} \end{array}$									
Description	Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTP instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual.The value of op2 defines the length of the effected										
Note	The EXTF "Notes on The EXTP	P instru ATOM P instru	uction m IC and E iction is	iust EXTe not a	be us ended availat	ed caref instructi le for ST	<sup>t</sup> ully (seeS ons"). F10X166 d	ection 4.6 levices.			
<b>Condition Flags</b>											
	E	Z	V	<u>с</u>	N						
	_										
	E	Not at	ffected.								
	Z	Not at	fected.								
	v C	Not at	fected.								
	N	Not at	ffected.								
Addressing Modes	Mnemonio	;		ļ	Forma	ıt		Bytes			
-	EXTP	Rwm,	#data <sub>2</sub>		DC :0	1##-m		2			
	EXTP	#pag,	#data <sub>2</sub>		D7 :01	##-0 pp	0:00pp	4			

86/185

EXTPR	Begin EX	Tend	ed Page	and R	egis	ter Sequence						
Syntax	EXTPR	op1,	op2									
Operation	$(\text{count}) \leftarrow (\text{op2}) \ [1 \le \text{op2} \le 4]$ Disable interrupts and Class A traps Data_Page = (op1) AND SFR_range = Extended DO WHILE ((count) \neq 0 AND Class_B_trap_condition \neq TRUE) Next Instruction (count) \leftarrow (count) - 1 END WHILE (count) = 0 Data_Page = (DPPx) AND SFR_range = Standard Enable interrupts and traps											
Description	Overrides and indire accesses being ma- of instruc interrupts ('mem') of sequence not deter value of of is derived of op2 de	Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. For any long ('mem') or indirect ([]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual. The value										
Note	The EXT "Notes on The EXT	PR ins ATOI PR ins	struction r MIC and struction i	nust b EXTen s not a	e use ded i vaila	ed carefully (see Section 4.6 instructions"). ble for ST10X166 devices.						
<b>Condition Flags</b>	_	_		_								
	E -	Z -	- V	C -	N -							
		Not	offootod									
	L 7	Not a	affected.									
	V	Not	affected.									
	С	Not a	affected.									
	N	Not a	affected.									
Addressing Modes	Mnemoni EXTPR EXTPR	c Rwm #pag	n, #data <sub>2</sub> g, #data <sub>2</sub>			Format Bytes DC :11##-m 2 D7 :11##-0 pp 0:00pp 4						

EXTS	Begin EX	(Tende	ed Segm	ent Seo	que	nce					
Syntax	EXTS	op1, (	op2								
Operation	(count) ← Disable ir Data_Sec DO WHIL Next Inst (count) ← END WH (count) = Data_Pac Enable in	$\leftarrow (op2) [1 \le op2 \le 4]$ interrupts and Class A traps $= gment = (op1)$ ILE ((count) $\ne 0$ AND Class_B_trap_condition $\ne$ TRUE) struction $\leftarrow (count) - 1$ HILE $= 0$ age = (DPPx) interrupts and traps									
Description	Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTS instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([]) address in an EXTS instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15-A0). The value of op2 defines the length of the effected instruction										
Note	The EXT "Notes or The EXT	S instru n ATON S instru	uction m IIC and E uction is i	ust be u EXTende not avai	useo ed i ilabl	d carefully (see Section 4.6 nstructions"). e for ST10X166 devices.					
<b>Condition Flags</b>											
	E	Z	V	C	N						
	_			_							
	E	Not a	ffected.								
	Z	Not a	ffected.								
	V										
	C	Not a	ffected.								
A 11	IN .	Not a	mected.								
Addressing Modes	Mnemoni	C	#data			Format Bytes					
	EXIS	TWIN #soc	, #uala <sub>2</sub> #data-			DC .00##-III 2					
	LAIO	<i>т</i> зсу,	mualag			$1.00\pi\pi^{-0}$ 35 00 4					

EXTSR	Begin EXTended Segment and Register Sequence										
Syntax	EXTSR	op1,	op2								
Operation	$(\operatorname{count}) \leftarrow (\operatorname{op2}) \ [1 \le \operatorname{op2} \le 4]$ Disable interrupts and Class A traps Data_Segment = (op1) AND SFR_range = Extended DO WHILE ((count) \neq 0 AND Class_B_trap_condition \neq TRUE) Next Instruction (count) \leftarrow (count) - 1 END WHILE (count) = 0 Data_Page = (DPPx) AND SFR_range = Standard Enable interrupts and traps										
Description	Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTSR instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([]) address in an EXTSR instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15- A0). The value of op2 defines the length of the effected instruc-										
Note	The EXTS 4.6 "Notes The EXTS	SR in s on A SR ins	struction ATOMIC a struction i	must k Ind EX s not a	be u Tend vaila	sed c led ins ble fo	arefully( structions r ST10X1	see ;"). 166 (	Section devices.		
Condition Flags	_			_							
	E	Z	V	<b>C</b>	Ν						
	F	- Not	- affected	-	-						
	7	Not	affected.								
	V	Not	affected.								
	C	Not	affected.								
	N	Not	affected.								
Addressing Modes	Mnemonio EXTSR	c Rwn	n, #data <sub>2</sub>			Form DC :1	at 0##-m	E	Bytes 2		
	EXTSR	#seg	g, #data <sub>2</sub>			D7 :1	0##-0 ss	00	4		

#### IDLE

IDLE	Enter Idle Mode									
Syntax	IDLE									
Operation	Enter Id	le Mod	е							
Description	This ins mode, th running. external accident tion.	This instruction causes the part to enter the idle mode. In this node, the CPU is powered down while the peripherals remain running. It remains powered down until a peripheral interrupt or external interrupt occurs. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.								
Condition Flags	F	7	v	C	N					
	-	-	-	-	-					
	E	Not	affected							
	Z	Not	affected							
	V	Not	affected							
	С	C Not affected.								
	Ν	Not	affected							
Addressing Modes	Mnemor IDLE	nic			F 8	ormat 7 78 87 87	Bytes 4			

JB	Relative	Jump	o if Bit S	et								
Syntax	JB	op1,	op2									
Operation	IF (op1) = (IP) ← (I ELSE Next Inst END IF	= 1 T⊢ P) + s ructio	IEN sign_exte	end (op	2)							
Data Types	BIT	ЗІТ										
Description	If the bit s the locat displacen number w in words. tion is the If the sp instruction	If the bit specified by op1 is set, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The displacement is a two's complement number which is sign extended and counts the relative distance in words. The value of the IP used in the target address calcula- tion is the address of the instruction following the JB instruction. If the specified bit is clear, the instruction following the JB instruction is executed.										
Condition Flags	E	7	V	C	N							
	-	-	• -	-	-							
	E	Not	affected									
	Z	Not	affected									
	V	V Not affected.										
	С	Not	affected									
	Ν	Not	affected									
Addressing Modes	Mnemoni JB	c bitac	ddr <sub>Q.q</sub> , ro	əl		Format 8A QQ rr q0	Bytes 4					

JB

JBC	Relative .	elative Jump if Bit Set and Clear Bit							
Syntax	JBC	op1,	op2						
Operation	IF (op1) = (op1) = 0 (IP) ← (IF ELSE Next Inst END IF	F (op1) = 1 THEN (op1) = 0 (IP) $\leftarrow$ (IP) + sign_extend (op2) ELSE Next Instruction END IF							
Data Types	BIT								
Description	If the bit specified by op1 is set, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The bit specified by op1 is cleared, allowing implementation of semaphore operations. The displacement is a two's complement number which is sign extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JBC instruction. If the specified bit was clear, the instruction following the JBC instruction is executed.								
Condition Flags	F	7	v	C	N				
	-	B	-	-	B				
	E	Not	affected						
	Z	Cont spec	tains loo cified bit	jical ne	gatio	n of the previous	state of the		
	V	Not	affected						
	С	Not	affected						
	Ν	Con	tains the	e previo	ous st	ate of the specif	ied bit.		
Addressing Modes	Mnemonio JBC	bitac	ddr <sub>Q.q</sub> , r	el		Format AA QQ rr q0	Bytes 4		



JMPA	Absolut	e Con	dition	al Jumj	C		
Syntax	JMPA	op1,	op2				
Operation	IF (op1) (IP) ← c w Next Ins END IF	= 1 T⊦ op2 structio	IEN m				
Description	If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and the instruction following the JMPA instruc- tion is executed normally.						
Condition Codes	See condition code Table 4.17 on page 45.						
Condition Flags	_	_		_			
	E	Z	V	C	N		
	E			Not affe	ected.		
	Z			Not affe	ected.		
	V			Not affe	ected.		
	C			Not affe	ected.		
	N			Not affe	ected.		
Addressing Modes	Mnemor JMPA	nic cc, c	caddr			Format EA c0 MM MM	Bytes 4

JMPI	Indirect C	condit	ional Ju	ımp			
Syntax	JMPI	op1,	op2				
Operation	IF (op1) = (IP) ← (o ELSE Next Instr END IF	1 THI p2) ructior	ΞN				
Description	If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and the instruction following the JMPI instruc- tion is executed normally.						
Condition Codes	See condi	tion co	ode Tabl	e 4.17	on p	age 45.	
Condition Flags	_			_			
	E	Z	V	С	Ν		
	-	-	-	-	-		
	E	Not a	ffected.				
	Z	Not a	ffected.				
	V	Not a	ffected.				
	С	Not a	ffected.				
	Ν	Not a	ffected.				
Addressing Modes	Mnemonio JMPI	с сс, [F	₹w <sub>n</sub> ]			Format 9C cn	Bytes 2

JMPR	Relative Con	ditiona	al Jump				
Syntax	JMPR op	1, op2					
Operation	$IF (op1) = 1 T$ $(IP) \leftarrow (IP) +$ $ELSE$ $Next Instruct\\END IF$	HEN sign_e ion	xtend (op2	2)			
Description	If the condition specified by op1 is met, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The displacement is a two's complement number which is sign extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JMPR instruction. If the specified condition is not met, program execution continues normally with the instruction following the JMPR instruction.						
<b>Condition Codes</b>	See condition	code 1	able 4.17	on p	age 45.		
Condition Flags	E         Z           -         -           E         -           Z         -           V         -           C         N	V -	C Not affec Not affec Not affec Not affec Not affec	N ted. ted. ted. ted. ted.			
Addressing Modes	Mnemonic JMPR cc,	rel			Format cD rr	Bytes 2	

#### JMPS

JMPS	Absolute	Inter-Se	egment	t Jump			
Syntax	JMPS	op1, op	)2				
Operation	$(CSP) \leftarrow o$ $(IP) \leftarrow op2$	op1 2					
Description	Branches op2 within	uncond the seg	itionally pment s	to the pecified	abso by c	olute address sp op1.	pecified by
Condition Flags	E	z	v	с	N	1	
	-	-	-	-	-		
	E	Not affe	ected.				
	Z	Not affe	ected.				
	V	Not affe	ected.				
	С	Not affe	ected.				
	Ν	Not affe	ected.				
Addressing Modes	Mnemonio	;			F	ormat	Bytes
	JMPS	seg, ca	ddr		F	A ss MM MM	4



JNB	Relative .	Jump i	if Bit Cl	ear			
Syntax	JNB	op1, d	op2				
Operation	IF (op1) = (IP) ← (IF ELSE Next Instr END IF	0 THE P) + się ruction	EN gn_exter	nd (op	2)		
Data Types	BIT						
Description	If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The displacement is a two's complement number which is sign extended and counts the relative distance in words. The value of the IP used in the target address calcula- tion is the address of the instruction following the JNB instruc- tion. If the specified bit is set, the instruction following the JNB instruction is executed						
Condition Flags	-	7	V	0	N		
	E -	-	-	-	N -		
	E	Not a	ffected.				
	Z	Not a	ffected.				
	V	Not a	ffected.				
	С	Not a	ffected.				
	Ν	Not a	ffected.				
Addressing Modes	Mnemonic JNB	; bitado	dr <sub>Q.q</sub> , re	I		Format 9A QQ rr q0	Bytes 4

97/185

JNBS	Relative .	e Jump if Bit Clear and Set Bit							
Syntax	JNBS	op1,	op2						
Operation	IF (op1) = (op1) = 1 (IP) ← (II ELSE Next Inst END IF	[op1) = 0 THEN o1) = 1 P) ← (IP) + sign_extend (op2) SE ext Instruction D IF							
Data Types	BIT								
Description	If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The bit specified by op1 is set, allowing implementation of semaphore operations. The displacement is a two's complement number which is sign extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JNBS instruction. If the specified bit was set, the instruction following the JNBS instruction is executed.								
Condition Flags	E	7	V	C	N				
	E	B	-	-	B				
	E 7	Not a	affected	nical ne	gatio	 of the previous	state of the		
	-	spec	cified bit		gauoi				
	V	Not	affected						
	С	Not	affected	•					
	Ν	Con	tains the	e previo	ous st	ate of the specifi	ed bit.		
Addressing Modes	Mnemonie JNBS	c bitac	ldr <sub>Q.q</sub> , r	el		Format BA QQ rr q0	Bytes 4		



MOV	Move Dat	a							
Syntax	MOV	op1, op2							
Operation	$(op1) \leftarrow (op1)$	op2)							
Data Types	WORD								
Description	Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.								
Condition Flags	F	7 V							
	*	* -	- *						
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.							
	Z	Set if the value of the source operand op2 equals zero. Cleared otherwise.							
	V	Not affected.							
	С	Not affected.							
	Ν	Set if the more op2 is set. Cle	st significa eared othe	nt bit of the sourc rwise.	e operand				
Addressing Modes	Mnemonio	C		Format	Bytes				
U	MOV	Rw <sub>n</sub> , Rw <sub>m</sub>		F0 nm	2				
	MOV	Rw <sub>n</sub> , #data <sub>4</sub>		E0 #n	2				
	MOV	reg, #data <sub>16</sub>		E6 RR ## ##	4				
	MOV	Rw <sub>n</sub> , [Rw <sub>m</sub> ]		A8 nm	2				
	MOV	Rw <sub>n</sub> , [Rw <sub>m</sub> +]		98 nm	2				
	MOV	[Rw <sub>m</sub> ], Rw <sub>n</sub>		B8 nm	2				
	MOV	[-Rw <sub>m</sub> ], Rw <sub>n</sub>		88 nm	2				
	MOV	[Rw <sub>n</sub> ], [Rw <sub>m</sub> ]		C8 nm	2				
	MOV	[Rw <sub>n</sub> +], [Rw <sub>m</sub>	]	D8 nm	2				
	MOV	[Rw <sub>n</sub> ], [Rw <sub>m</sub> +	]	E8 nm	2				
	MOV	Rw <sub>n</sub> , [Rw <sub>m</sub> +#	data <sub>16</sub> ]	D4 nm ## ##	4				
	MOV	[Rw <sub>m</sub> +#data <sub>1</sub>	<sub>6</sub> ], Rw <sub>n</sub>	C4 nm ## ##	4				
	MOV	[Rw <sub>n</sub> ], mem		84 0n MM MM	4				
	MOV	mem, [Rw <sub>n</sub> ]		94 0n MM MM	4				
	MOV	reg, mem		F2 RR MM MM	4				
	MOV	mem, reg		F6 RR MM MM	4				

#### MOVB

MOVB	Move Da	ata								
Syntax	MOVB	op1, op2								
Operation	(op1) ←	(op1) ← (op2)								
Data Types	BYTE									
Description	Moves the location of the location of the location of the second	Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.								
Condition Flags	_									
	E	Z V C	N *							
		····								
	E	Set if the value of op2 negative number. Cleathe end of a table.	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.							
	Z	Set if the value of the source operand op2 equals zero. Cleared otherwise.								
	V	Not affected.								
	С	Not affected.								
	Ν	Set if the most signific op2 is set. Cleared oth	ant bit of the sourd erwise.	ce operand						
Addressing Modes	Mnemon	ic	Format	Bytes						
U	MOVB	Rb <sub>n</sub> , Rb <sub>m</sub>	F1 nm	2						
	MOVB	Rb <sub>n</sub> , #data <sub>4</sub>	E1 #n	2						
	MOVB	reg, #data <sub>16</sub>	E7 RR ## ##	4						
	MOVB	Rb <sub>n</sub> , [Rw <sub>m</sub> ]	A9 nm	2						
	MOVB	Rb <sub>n</sub> , [Rw <sub>m</sub> +]	99 nm	2						
	MOVB	[Rw <sub>m</sub> ], Rb <sub>n</sub>	B9 nm	2						
	MOVB	[-Rw <sub>m</sub> ], Rb <sub>n</sub>	89 nm	2						
	MOVB	[Rw <sub>n</sub> ], [Rw <sub>m</sub> ]	C9 nm	2						
	MOVB	[Rw <sub>n</sub> +], [Rw <sub>m</sub> ]	D9 nm	2						
	MOVB	[Rw <sub>n</sub> ], [Rw <sub>m</sub> +]	E9 nm	2						
	MOVB	Rb <sub>n</sub> , [Rw <sub>m</sub> +#data <sub>16</sub> ]	F4 nm ## ##	4						
	MOVB	[Rw <sub>m</sub> +#data <sub>16</sub> ], Rb <sub>n</sub>	E4 nm ## ##	4						
	MOVB	[Rw <sub>n</sub> ], mem	A4 0n MM MM	4						
	MOVB	mem, [Rw <sub>n</sub> ]	B4 0n MM MM	4						
	MOVB	reg, mem	F3 RR MM MM	4						
	MOVB	mem, reg	F7 RR MM MM	4						

MOVBS	Move Byte Sign Extend							
Syntax	MOVBS	MOVBS op1, op2						
Operation	$\begin{array}{l} (\text{low byte op1}) \leftarrow (\text{op2}) \\ \text{IF } (\text{op2}_7) = 1 \text{ THEN} \\ (\text{high byte op1}) \leftarrow \text{FF}_{\text{H}} \\ \text{ELSE} \\ (\text{high byte op1}) \leftarrow 00_{\text{H}} \\ \text{END IF} \end{array}$							
Data Types	WORD, B	YTE						
Description	Moves and sign extends the contents of the source byte specified by op2 to the word location specified by the destina- tion operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.							
Condition Flags	<b>E</b>	Z V C N * *						
	E	Always cleared.						
	Z	Set if the value of the s zero. Cleared otherwise.	ource operand op2 equals					
	V	Not affected.						
	С	Not affected.						
	Ν	Set if the most significant op2 is set. Cleared other	It bit of the source operand wise.					
Addressing Modes	Mnemonio MOVBS MOVBS MOVBS	c Rw <sub>n</sub> , Rb <sub>m</sub> reg, mem mem, reg	FormatBytesD0 mn2D2 RR MM MM4D5 RR MM MM4					

#### MOVBZ

MOVBZ	Move Byte Zero Extend							
Syntax	MOVBZ	op1,	op2					
Operation	(low byte (high byte	(low byte op1) $\leftarrow$ (op2) (high byte op1) $\leftarrow$ 00 <sub>H</sub>						
Data Types	WORD, B	YTE						
Description	Moves and zero extends the contents of the source byte specified by op2 to the word location specified by the destina- tion operand op1. The contents of the moved data is examined, and the condition codes are updated accordingly.							
Condition Flags								
	0	*	-	-	0			
	E	Alwa	iys clea	red.				
	Z	Set zero	if the va . Cleare	alue of d other	the s wise.	ource operand	op2 equals	
	V	Not a	affected	-				
	С	Not a	affected					
	Ν	Alwa	iys clea	red.				
Addressing Modes	Mnemonie MOVBZ MOVBZ	Rw <sub>n</sub> reg,	, Rb <sub>m</sub> mem			Format C0 mn C2 RR MM MM C5 RR MM MM	Bytes 2 4 4	

MUL	Signed Multiplication							
Syntax	MUL	op1, op2						
Operation	$(MD) \leftarrow (C$	op1) * (op2)						
Data Types	WORD							
Description	Performs a 16-bit by 16-bit signed multiplication using the two words specified by operands op1 and op2 respectively. The signed 32-bit result is placed in the MD register.							
Condition Flags	_		•					
	Е 0	2 V * S	0	<b>N</b> 0				
	E	Always clear	ed.					
	Z	Set if the res	ult equals	s zero. Cleared othe	rwise.			
	V	This bit is se word data typ	t if the reacted to the	sult cannot be represed otherwise.	sented in a			
	С	Always clear	ed.					
	Ν	Set if the most significant bit of the result is set. Cleared otherwise.						
Addressing Modes	Mnemonic	;	Format	Bytes				
	MUL	Rw <sub>n</sub> , Rw <sub>m</sub>		0B nm	2			

#### MULU

MULU	Unsigned	Mult	iplicati	on				
Syntax	MULU	op1,	op2					
Operation	$(MD) \leftarrow (c$	op1) *	(op2)					
Data Types	WORD							
Description	Performs two words unsigned 3	a 16- spec 32-bit	bit by f ified by result is	16-bit u operan s placeo	nsigne ds op1 d in the	ed multiplication and op2 respe MD register.	n using the ectively. The	
Condition Flags	-	7	V	•				
	<b>E</b>	*	v S	0	0			
	E	Alwa	ys clear	ed.		_		
	Z	Set if the result equals zero. Cleared otherwise.						
	V	This bit is set if the result cannot be represented in a word data type. Cleared otherwise.						
	С	Always cleared.						
	Ν	Set if the most significant bit of the result is set. Cleared otherwise.						
Addressing Modes	Mnemonic MULU	nic Format Bytes Rw <sub>n</sub> , Rw <sub>m</sub> 1foB nm 2						



NEG	Integer T	wo's Co	omplerr	nent				
Syntax	NEG	op1						
Operation	$(op1) \leftarrow 0$	- (op1)						
Data Types	WORD							
Description	Performs specified l	a bina by op1.	ry 2's The res	comple sult is th	ement nen sto	of the so pred in op1	ource operand	
Condition Flags	E *	<b>Z</b>	<b>V</b>	<b>C</b> S	N *			
	E	Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.						
	Z	Set if result equals zero. Cleared otherwise.						
	V	Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.						
	С	Set if a borrow is generated. Cleared otherwise.						
	Ν	Set if Cleare	the mo d other	ost sign wise.	ificant	bit of the	e result is set.	
Addressing Modes	Mnemonio NEG	Rw <sub>n</sub>			Fo 81	ormat n0	Bytes 2	

#### NEGB

NEGB	Integer Tv	vo's Co	mplem	ent			
Syntax	NEGB	op1					
Operation	(op1) ← 0	- (op1)					
Data Types	BYTE						
Description	Performs specified b	a bina by op1.	ry 2's The res	comple sult is t	ement hen sto	of the source pred in op1.	e operand
Condition Flags	E *	<b>Z</b>	<b>V</b>	C S	<b>N</b> *		
	E	Set if the value of op1 represents the lowest possibl negative number. Cleared otherwise. Used to signathe end of a table.					
	Z	Set if result equals zero. Cleared otherwise.					
	V	Set if an arithmetic underflow occurred, i.e. the resu cannot be represented in the specified data type Cleared otherwise.					
	С	Set if a borrow is generated. Cleared otherwise.					
	Ν	Set if Cleare	the mo d other	st sigr wise.	nificant	t bit of the res	sult is set.
Addressing Modes	Mnemonic NEGB	; Rb <sub>n</sub>			Fc A1	ormat 1 n0	Bytes 2



NOP	No Ope	eration				
Syntax	NOP					
Operation	No Ope	eration				
Description	This ins operation	structior	n causes es no cl	s a null nange i	operati n the st	( 
Condition Flags	_	_	.,	•		
	E	Ζ	V	C	N	
	-	-	-	-	-	
	Е	Not	affected			
	Z	Not	affected			
	V	Not	affected			
	С	Not	affected			
	Ν	Not	affected			
Addressing Modes	Mnemo NOP	onic			F C	

OR

OR	Logical O	R							
Syntax	OR	op1,	op2						
Operation	$(op1) \leftarrow (a)$	op1) \	⁄ (op2)						
Data Types	WORD								
Description	Performs by op2 and is then sto	a bitv d the ored ir	vise logi destinat n op1.	cal OR tion ope	of the erand s	source operand pecified by op1.	d specified The result		
Condition Flags									
	E	Z	V	C	N	٦			
	*	*	0	0	*				
	Е	Set i nega the e	f the val ative nur end of a	ue of oj nber. C table.	o2 repr leared	esents the lowe otherwise. Use	st possible d to signal		
	Z	Set if result equals zero. Cleared otherwise.							
	V	Always cleared.							
	С	Always cleared.							
	Ν	Set Clea	if the m red othe	nost sig erwise.	gnifican	t bit of the res	sult is set.		
Addressing Modes	Mnemonio	)			F	ormat	Bytes		
	OR	Rwn	, Rw <sub>m</sub>		7	0 nm	2		
	OR	Rwn	, [Rw <sub>i</sub> ]		7	8 n:10ii	2		
	OR	Rwn	, [Rw <sub>i</sub> +]		7	8 n:11ii	2		
	OR	Rwn	, #data <sub>3</sub>		7	8 n:0###	2		
	OR	reg,	#data <sub>16</sub>		7	6 RR ## ##	4		
	OR	reg,	mem		72	2 RR MM MM	4		
	OR	mem	n, reg		74	4 RR MM MM	4		


ORB	Logical C	R						
Syntax	ORB	op1,	op2					
Operation	$(op1) \leftarrow (op1)$	op1) \	√ (op2)					
Data Types	BYTE							
Description	Performs a bitwise logical OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.							
Condition Flags	_	_		-				
	E	Z	V	С	N	7		
	*	×	0	0	*			
	Е	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.						
	Z	Set i	f result	equals	zero. C	leared otherwis	e.	
	V	Alwa	ays clea	red.				
	С	Alwa	ays clea	red.				
	Ν	Set Clea	if the r red oth	nost sią erwise.	gnificar	nt bit of the rea	sult is set.	
Addressing Modes	Mnemonio	5			F	ormat	Bytes	
	ORB	Rb <sub>n</sub> ,	Rb <sub>m</sub>		7	1 nm	2	
	ORB	Rb <sub>n</sub> ,	[Rw <sub>i</sub> ]		7	9 n:10ii	2	
	ORB	Rb <sub>n</sub> ,	[Rw <sub>i</sub> +]		7	9 n:11ii	2	
	ORB	Rb <sub>n</sub> ,	#data3		7	9 n:0###	2	
	ORB	reg,	#data <sub>16</sub>	5	7	7 RR ## ##	4	
	ORB	reg,	mem		7	3 RR MM MM	4	
	ORB	mem	n, reg		7	5 RR MM MM	4	

#### PCALL

PCALL	Push Wor	rd and Call Subroutine Absolute							
Syntax	PCALL	op1,	op2						
Operation	$(tmp) \leftarrow (o)$ $(SP) \leftarrow (S)$ $((SP)) \leftarrow (O)$ $(SP) \leftarrow (S)$ $((SP)) \leftarrow (O)$ $((SP)) \leftarrow (O)$ $(IP) \leftarrow Op)$	$\begin{array}{l} (\text{tmp}) \leftarrow (\text{op1}) \\ (\text{SP}) \leftarrow (\text{SP}) - 2 \\ ((\text{SP})) \leftarrow (\text{tmp}) \\ (\text{SP}) \leftarrow (\text{SP}) - 2 \\ ((\text{SP})) \leftarrow (\text{IP}) \\ (\text{IP}) \leftarrow \text{op2} \end{array}$							
Data Types	WORD	WORD							
Description	Pushes the word specified by operand op1 and the value of the instruction pointer, IP, onto the system stack, and branches to the absolute memory location specified by the second operand op2. Because IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address of the calling routine.								
Condition Flags	_	_							
	E *	Z *	-	C -	N *				
	E	Set i sents other	f the v s the lc rwise. l	alue of west po Jsed to	the pu ossible signal	ushed operand negative numbe the end of a tab	op1 repre- er. Cleared le.		
	Z	Set i zero.	f the v Cleare	alue of ed other	the pu wise.	ished operand o	op1 equals		
	V	Not a	affected	J.					
	С	Not a	affected	J.					
	Ν	Set i op1 i	f the m s set. (	lost sigr Cleared	nificant otherv	bit of the pushe	d operand		
Addressing Modes	Mnemonic CALL	reg, (	caddr		F	Format E2 RR MM MM	Bytes 4		



POP	Pop Word	op Word from System Stack							
Syntax	POP	op1							
Operation	$\begin{array}{l} (tmp) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \\ (op1) \leftarrow (tmp) \end{array}$								
Data Types	WORD								
Description	Pops one word from the system stack specified by the Stack Pointer into the operand specified by op1. The Stack Pointer is then incremented by two.								
Condition Flags	E *	Z V C N * *							
	E	Set if the value of the p lowest possible negative Used to signal the end o	oopped word repronumber. Cleared f a table.	esents the otherwise.					
	Z	Set if the value of the Cleared otherwise.	popped word eq	uals zero.					
	V	Not affected.							
	С	Not affected.							
	Ν	Set if the most significant bit of the popped word is set. Cleared otherwise.							
Addressing Modes	Mnemonic POP	reg	Format FC RR	Bytes 2					

#### PRIOR

PRIOR	Prioritize	Register						
Syntax	PRIOR	op1, op2	2					
Operation	$\begin{array}{l} (tmp) \leftarrow (op2) \\ (count) \leftarrow 0 \\ DO \ WHILE \ (tmp_{15}) \neq 1 \ AND \ (count) \neq 15 \ AND \ (op2) \neq 0 \\ (tmp_n) \leftarrow (tmp_{n-}1) \\ (count) \leftarrow (count) - 1 \\ END \ WHILE \\ (op1) \leftarrow (count) \end{array}$							
Data Types	WORD							
Description	This instruction stores a count value in the word operand specified by op1 indicating the number of single bit shifts required to normalize the operand op2 so that its MSB is equal to one. If the source operand op2 equals zero, a zero is written to operand op1 and the zero flag is set. Otherwise the zero flag is cleared.							
Condition Flags	_							
	Е 0	× 0		, )	<b>N</b>	]		
	E	Always o	leared.					
	Z	Set if the otherwis	e sourc e.	e ope	erand	op2 equals ze	ro. Cleared	
	V	Always cleared.						
	С	Always o	leared.					
	Ν	Always o	leared.					
Addressing Modes	Mnemonic PRIOR	; Rw <sub>n</sub> , Rw	'n		Fo 2E	ormat 3 nm	Bytes 2	



PUSH	Push Wo	Word on System Stack							
Syntax	PUSH	op1							
Operation	$(tmp) \leftarrow (orgonal (SP)) \leftarrow (SP)$ $((SP)) \leftarrow (SP)$	$\begin{array}{l} (tmp) \leftarrow (op1) \\ (SP) \leftarrow (SP) - 2 \\ ((SP)) \leftarrow (tmp) \end{array}$							
Data Types	WORD								
Description	Moves the word specified by operand op1 to the location in the internal system stack specified by the Stack Pointer, after the Stack Pointer has been decremented by two.								
Condition Flags	E *	Z V C N * *		aconto the					
	C	lowest possible negative Used to signal the end of	number. Cleared	otherwise.					
	Z	Set if the value of the Cleared otherwise.	pushed word ec	quals zero.					
	V	Not affected.							
	С	Not affected.							
	Ν	Set if the most significates set. Cleared otherwise.	ant bit of the push	ed word is					
Addressing Modes	Mnemonio PUSH	reg	Format EC RR	Bytes 2					

#### **PWRDN**

PWRDN	Enter Pov	ver D	own Mo	ode					
Syntax	PWRDN	PWRDN							
Operation	Enter Pow	Enter Power Down Mode							
Description	This instruction causes the part to enter the power down mode. In this mode, all peripherals and the CPU are powered down until the part is externally reset. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction. To further control the action of this instruction, the PWRDN instruction is only enabled when the non-maskable interrupt pin ( $\overline{NMI}$ ) is in the low state. Otherwise, this instruction has no effect.								
<b>Condition Flags</b>									
	E -	Z -	V -	C -	N -				
	E	Not	affected						
	Z	Not	affected						
	V	Not	affected						
	С	Not	affected						
	Ν	Not	affected						
Addressing Modes	Mnemonic PWRDN	)				Format 97 68 97 97	Bytes 4		

RET	Return fr	om S	ubrouti	ne				
Syntax	RET							
Operation	$\begin{array}{l} (IP) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \end{array}$							
Description	Returns from a subroutine. The IP is popped from the system stack. Execution resumes at the instruction following the CALL instruction in the calling routine.							
Condition Flags	EZVCN							
	-	-	-	-	-	]		
	E	Not	affected.					
	Z	Not	affected.					
	V	Not	affected.					
	С	Not	affected.					
	Ν	Not	affected.					
Addressing Modes	Mnemonio RET	C			Fo C	ormat B 00	Bytes 2	

RETI	Return fro	om Interi	rupt Ro	outin	e		
Syntax	RETI						
Operation	$\begin{array}{l} (IP) \leftarrow ((S\\ (SP) \leftarrow (S\\ IF \ (SYSCG\\ (CSP) \leftarrow (SP) \leftarrow (S$	P)) P) + 2 DN.SGT[ ((SP)) SP) + 2 ((SP)) P) + 2	DIS=0)	THE	N		
Description	Returns fr popped of tion which restored a popped if SGTDIS b	om an ir f the syst had bee after the segmen it in the \$	nterrup tem sta en inter PSW tation SYSCC	t rou ack. E rrupte has is er DN re	tine. Execu ed. Ti been nable giste	The PSW, IP tion resumes he previous s popped. The d. This is inc r.	and CSP are at the instruc- system state is CSP is only dicated by the
Condition Flags	Е	Z \	/	с	N		
	S	S S	6	S	S		
	E	Restore	d from	the F	PSW	popped from	stack.
	Z	Restore	d from	the F	PSW	popped from	stack.
	V	Restore	d from	the F	PSW	popped from	stack.
	С	Restore	d from	the F	PSW	popped from	stack.
	Ν	Restore	d from	the F	PSW	popped from	stack.
Addressing Modes	Mnemonic RETI	;				Format FB 88	Bytes 2



RETP	Return fro	rom Subroutine and Pop Word							
Syntax	RETP	op1							
Operation	$\begin{array}{l} (IP) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \\ (tmp) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \\ (op1) \leftarrow (tmp) \end{array}$								
Data Types	WORD								
Description	Returns from a subroutine. The IP is first popped from the system stack and then the next word is popped from the system stack into the operand specified by op1. Execution resumes at the instruction following the CALL instruction in the calling routine.								
Condition Flags	<b>E</b>	Z V C N							
	_								
	E	Set if the value of the work represents the lowest Cleared otherwise. Used	ord popped into op possible negative I to signal the end	erand op1 e number. of a table.					
	Z	Set if the value of the wo equals zero. Cleared oth	ord popped into op erwise.	erand op1					
	V	Not affected.							
	С	Not affected.							
	Ν	Set if the most significan operand op1 is set. Clear	it bit of the word pared otherwise.	opped into					
Addressing Modes	Mnemonio RETP	reg	Format EB RR	Bytes 2					

### RETS

RETS	Return from Inter-Segment Subroutine							
Syntax	RETS							
Operation	$\begin{array}{l} (IP) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \\ (CSP) \leftarrow ((SP)) \\ (SP) \leftarrow (SP) + 2 \end{array}$							
Description	Returns from an inter-segment subroutine. The IP and CSP are popped from the system stack. Execution resumes at the instruction following the CALLS instruction in the calling routine.							
Condition Flags	Е	v	С	N				
	-	-	-	-	-	]		
	E	Not	affected					
	Z	Not	affected					
	V	Not	affected					
	C Not affected.							
	Ν	Not	affected					
Addressing Mode	Mnemon RETS	ic			Fo D	ormat B 00	Bytes 2	



ROL	Rotate Le	eft							
Syntax	ROL	op1,	op2						
Operation	$\begin{array}{l} (\text{count}) \leftarrow (\text{op2}) \\ (C) \leftarrow 0 \\ \text{DO WHILE (count)} \neq 0 \\ (C) \leftarrow (\text{op1}_{15}) \\ (\text{op1}_n) \leftarrow (\text{op1}_{n-1}) \ [n=115] \\ (\text{op1}_0) \leftarrow (C) \\ (\text{count}) \leftarrow (\text{count}) - 1 \\ \text{END WHILE} \end{array}$								
Data Types	WORD	WORD							
Description	Rotates the destination word operand op1 left by as many times as specified by the source operand op2. Bit 15 is rotated into Bit 0 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.								
Condition Flags	-	7	V	0	N				
	Е 0	*	<b>v</b> 0	с S	N *				
	E	Alwa	iys clear	ed.					
	Z	Set i	f result e	equals :	zero. Cl	eared otherwis	e.		
	V	Alwa	iys clear	ed.					
	С	The shifte	carry fied out of	ag is f op1. C	set acc cleared	cording to the for a rotate cou	last MSB int of zero.		
	Ν	Set Clea	if the m red othe	nost sig erwise.	nificant	bit of the res	sult is set.		
Addressing Modes	Mnemonio	C				ormat	Bytes		
	ROL	Rw <sub>n</sub>	, Rw <sub>m</sub>		00	C nm	2		
	KOL	KWn	, #data⊿		1(	;#n	.,		

ROR	Rotate Ri	ight							
Syntax	ROR	op1,	op2						
Operation	$\begin{array}{l} (\text{count}) \leftarrow (\text{op2}) \\ (C) \leftarrow 0 \\ (V) \leftarrow 0 \\ \text{DO WHILE (count)} \neq 0 \\ (V) \leftarrow (V) \lor (C) \\ (C) \leftarrow (\text{op1}_0) \\ (\text{op1}_n) \leftarrow (\text{op1}_{n+1})  [n=014] \\ (\text{op1}_{15}) \leftarrow (C) \\ (\text{count}) \leftarrow (\text{count}) - 1 \\ \text{END WHILE} \\ \end{array}$								
Data Types	WORD								
Description	Rotates the destination word operand op1 right by as many times as specified by the source operand op2. Bit 0 is rotated into Bit 15 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.								
Condition Flags	-	-		•					
	<b>E</b>	*	S S	S	*				
	E	Alwa	vs clear	ed.		l			
	Z	Set if	result e	equals z	zero. Cl	eared otherwis	e.		
	V	Set i shifte coun	f in any ed out o t of zero	/ cycle of the	of the carry fl	rotate operation ag. Cleared for	on a '1' is or a rotate		
	С	The o	carry fla f op1. C	g is set leared	accord for a ro	ing to the last L tate count of ze	.SB shifted ero.		
	Ν	Set i Clear	f the m red othe	iost sig rwise.	Inificant	t bit of the res	sult is set.		
Addressing Modes	Mnemonic	;			Fo	ormat	Bytes		
	ROR	Rw <sub>n</sub> ,	Rw <sub>m</sub>		20	Cnm	2		
	ROR	Rw <sub>n</sub> ,	#data <sub>4</sub>		30	C #n	2		



SCXT	Switch C	ontex	ĸt					
Syntax	SCXT	op1,	, op2					
Operation	$\begin{array}{l} (tmp1) \leftarrow \\ (tmp2) \leftarrow \\ (SP) \leftarrow (S) \\ ((SP)) \leftarrow \\ (op1) \leftarrow (S) \end{array}$	(op1) (op2) SP) - 2 (tmp1 tmp2)	) 2 1)					
Description	Used to s push and by the fin register is operand,	witch load rst op s ther op2.	context operatio perand, loaded	s for an on. The op1, a with tl	ny reg conte re pu ne val	gister ents Isheo lue s	: Switching of of the registed onto the sepecified by the	context is a er specified stack. That the second
Condition Flags								
U	E	Ζ	V	С	Ν			
	-	-	-	-	-			
	E	Not	affected					
	Z	Not	affected					
	V	Not	affected					
	С	Not	affected					
	Ν	Not	affected					
Addressing Modes	Mnemoni SCXT SCXT	c reg, reg,	#data <sub>16</sub> mem	i		Forr C6 F D6 F	nat RR ## ## RR MM MM	Bytes 4 4

SHL	Shift Left							
Syntax	SHL op1, op2							
Operation	$\begin{array}{l} (\text{count}) \leftarrow (\text{op2}) \\ (C) \leftarrow 0 \\ \text{DO WHILE (count)} \neq 0 \\ (C) \leftarrow (\text{op1}_{15}) \\ (\text{op1}_n) \leftarrow (\text{op1}_{n-1}) \ [n=115] \\ (\text{op1}_0) \leftarrow 0 \\ (\text{count}) \leftarrow (\text{count}) - 1 \\ \text{END WHILE} \end{array}$							
Data Types	WORD							
Description	Shifts the destination word operand op1 left by as many times as specified by the source operand op2. The least significant bits of the result are filled with zeros accordingly. The MSB is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.							
Condition Flags	F	7	v	С	N			
	0	*	0	S	*	]		
	E	Alwa	ivs clea	red.		_		
	Z	Set i	f result	equals :	zero. C	leared otherwis	e.	
	V	Alwa	iys cleai	red.				
	С	The carry flag is set according to the last MSB shifted out of op1. Cleared for a shift count of zero.						
	Ν	Set Clea	if the n red othe	nost sig erwise.	gnifican	t bit of the rea	sult is set.	
Addressing Modes	Mnemonic SHL SHL	ic Format Bytes $Rw_n, Rw_m$ 4C nm 2 $Rw_n, #data_4$ 5C #n 2						



SHR	Shift Rig	nt				
Syntax	SHR	op1, op2				
Operation	$\begin{array}{l} (\text{count}) \leftarrow \\ (C) \leftarrow 0 \\ (V) \leftarrow 0 \\ \text{DO WHIL} \\ (V) \leftarrow (C) \\ (C) \leftarrow (op \\ (op 1_n) \leftarrow \\ (op 1_{15}) \leftarrow \\ (\text{count}) \leftarrow \\ \text{END WHI} \end{array}$	(op2) E (count) ≠ 0 ) ∨ (V) o1 <sub>0</sub> ) (op1 <sub>n+1</sub> ) [n=0. - 0 - (count) - 1 LE	14]			
Data Types	WORD					
Description	Shifts the destination word operand op1 right by as many times as specified by the source operand op2. The most significant bits of the result are filled with zeros accordingly. Since the bits shifted out effectively represent the remainder, the Overflow flag is used instead as a Rounding flag. This flag together with the Carry flag helps the user to determine whether the remainder bits lost were greater than, less than or equal to one half an LSB. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least signifi- cant 4 bits are used.					
Condition Flags	Е	z v	C N			
	0	* S	S *			
	E	Always cleared	d.			
	Z	Set if result eq	juals zero.	Cleared otherwise	ə.	
	V	Set if in any cy out of the carry	vcle of the s y flag. Clea	shift operation a '1 ared for a shift cou	' is shifted int of zero.	
	С	The carry flag out of op1. Cle	is set acco eared for a	rding to the last L shift count of zero	SB shifted o.	
	Ν	Set if the mo Cleared otherv	st significa wise.	ant bit of the res	sult is set.	
Addressing Modes	Mnemonio	2		Format	Bytes	
	SHK	Kw <sub>n</sub> , Kw <sub>m</sub> Rw #data		6C nm 7C #p	2	
		n w <sub>n</sub> , #uala <sub>4</sub>			۷	

### SRST

SRST	Soft	ware	Rese	et				
Syntax	SRS	SRST						
Operation	Software Reset							
Description	This instruction is used to perform a software reset. A software reset has the same effect on the microcontroller as an exter- nally applied hardware reset. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.							
Condition Flags	E		Z	V	C	N		
	E Z V C N		Alwa Alwa Alwa Alwa Alwa	ays clea ays clea ays clea ays clea ays clea ays clea	ured. ured. ured. ured. ured.	0		
Addressing Modes	Mner	nonic T	;				Format B7 48 B7 B7	Bytes 4



**Syntax** 

Operation

Description

Service Watchdog Timer

SRVWDT

Service Watchdog Timer

This instruction services the Watchdog Timer. It reloads the high order byte of the Watchdog Timer with a preset value and clears the low byte on every occurrence. Once this instruction has been executed, the watchdog timer cannot be disabled. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

Con	dition	Flags
-----	--------	-------

	E	Z	V	С	N		
	-	-	-	-	-		
	Е	Not	affected				
	Z	Not	affected				
	V	Not	affected				
	С	Not	affected				
	Ν	Not	affected				
Addressing Modes	Mnemo SRVWI	onic DT			Fo A	ormat 7 58 A7 A7	Bytes 4

SUB	Integer S	ubtraction				
Syntax	SUB	op1, op2				
Operation	$(op1) \leftarrow (a)$	op1) - (op2)				
Data Types	WORD					
Description	Performs operand specified b	a 2's complement binar specified by op2 from by op1. The result is then	y subtraction of t the destination stored in op1.	he source operand		
Condition Flags						
	E Z	* * S *				
	E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.					
	Z	Set if result equals zero.	Cleared otherwise	е.		
	V	Set if an arithmetic under cannot be represented Cleared otherwise.	rflow occurred, i.e in the specified	. the result data type.		
	С	Set if a borrow is genera	ited. Cleared other	rwise.		
	Ν	Set if the most signific Cleared otherwise.	ant bit of the res	sult is set.		
Addressing Modes	Mnemonic	;	Format	Bytes		
	SUB	Rw <sub>n</sub> , Rw <sub>m</sub>	20 nm	2		
	SUB	Rw <sub>n</sub> , [Rw <sub>i</sub> ]	28 n:10ii	2		
	SUB	Rw <sub>n</sub> , [Rw <sub>i</sub> +]	28 n:11ii	2		
	SUB	Rw <sub>n</sub> , #data <sub>3</sub>	28 n:0###	2		
	SUB	reg, #data <sub>16</sub>	26 RR ## ##	4		
	SUB	reg, mem	22 RR MM MM	4		
	300	inem, ieg		4		



SUBB	Integer S	ubtraction							
Syntax	SUBB op1, op2								
Operation	$(op1) \leftarrow (op1)$	op1) - (op2)							
Data Types	BYTE								
Description	Performs operand specified l	a 2's compl specified b by op1. The	lement y op2 result is	binary from s then s	subtrac the de tored in	etion of t estination op1.	the source o operand		
Condition Flags	Condit	ion Elogo	E	7	v	c	NI		
	Condit	ion riags	*	*	V *	S	*		
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.							
	Z	Set if result equals zero. Cleared otherwise.							
	V	Set if an arithmetic underflow occurred, ie. the result cannot be represented in the specified data type. Cleared otherwise.							
	С	Set if a borr	ow is g	enerate	ed. Clea	red othe	rwise.		
	Ν	Set if the i Cleared oth	most si erwise.	ignificar	nt bit of	f the rea	sult is set.		
Addressing Modes	Mnemonia SUBB SUBB SUBB SUBB SUBB	c Rb <sub>n</sub> , Rb <sub>m</sub> Rb <sub>n</sub> , [Rw <sub>i</sub> ] Rb <sub>n</sub> , [Rw <sub>i</sub> +] Rb <sub>n</sub> , #data <sub>10</sub> reg, #data <sub>10</sub>	6	F 2 2 2 2 2 2 2 2 2	Format 21 nm 29 n:10ii 29 n:11ii 29 n:0## 27 RR #i 23 RR M	# # ## IM MM	Bytes 2 2 2 2 4 4		
	SUBB	mem, reg		2	5 RR M	IM MM	4		

### SUBC

SUBC	Integer St	ubtraction v	with Car	ry				
Synta	SUBC op1, op2							
Operation	$(op1) \leftarrow (c$	op1) - (op2)	- (C)					
Data Types	WORD							
Description	Performs operand s from the o then store multiple pr	a 2's comp pecified by c destination ed in op1. 7 recision arith	lement I pp2 and to perand This inst imetic.	binary the pre speci ructior	subtract eviously ( fied by ( can be	tion of t generate op1. The e used t	he source ed carry bit e result is to perform	
Condition Flags	Conditi	on Florio	F	7	V	0	N	
	Conditi	on Flags	*	S	V *	S	N *	
	E	Set if the vanegative nut the end of a	alue of op Imber. C a table.	p2 rep leared	resents t I otherwi	he lowe: se. Use	st possible d to signal	
	Z	Set if result set. Cleared	equals d otherw	zero a vise.	nd the p	revious	Z flag was	
	V	Set if an ari cannot be Cleared oth	thmetic represe erwise.	underf nted i	low occu n the sp	urred, i.e becified	. the result data type.	
	С	Set if a borr	ow is ge	enerate	ed. Clear	ed other	rwise.	
	Ν	Set if the Cleared oth	most sig ierwise.	gnifica	nt bit of	the res	sult is set.	
Addressing Modes	Mnemonic	;		F	Format		Bytes	
	SUBC	Rw <sub>n</sub> , Rw <sub>m</sub>		3	30 nm		2	
	SUBC	Rw <sub>n</sub> , [Rw <sub>i</sub> ]	_	3	38 n:10ii		2	
	SUBC	Rw <sub>n</sub> , [Rw <sub>i</sub> +]		3	38 n:11ii		2	
	SUBC	Rw <sub>n</sub> , #data	3	3	38 n:0##;	#	2	
	SUBC	reg, #data <sub>1</sub>	6	3	36 RR ##	# ## • • • • • •	4	
	SUBC	reg, mem			32 KK M	IVI IVIM Na nana	4	
	JUDU	mem, reg		Ċ	04 rr IVI		4	



# **SUBCB**

57

# Integer Subtraction with Carry

Syntax	SUBCB	op1,	op2				
Operation	(op1) ← (	op1) ·	- (op2) -	(C)			
Data Types	BYTE						
Description	Performs a 2's complement binary subtraction of the source operand specified by op2 and the previously generated carry bit from the destination operand specified by op1. The result is then stored in op1. This instruction can be used to perform multiple precision arithmetic.						the source ed carry bit ne result is to perform
Condition Flags	F	7	v	C	Ν		
	*	*	*	S	*	7	
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signative end of a table.					est possible ed to signal
	Z	Set if result equals zero. Cleared otherwise.					
	V	Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.					
	С	Set	if a borr	ow is g	enerate	ed. Cleared othe	erwise.
	Ν	Set Clea	if the r ared oth	nost si erwise.	gnifica	nt bit of the re	sult is set.
Addressing Modes	Mnemonie SUBCB SUBCB SUBCB SUBCB SUBCB SUBCB	c Rb <sub>n</sub> , Rb <sub>n</sub> , Rb <sub>n</sub> , reg, reg, reg,	, Rb <sub>m</sub> , [Rw <sub>i</sub> ] , [Rw <sub>i</sub> +] , #data <sub>3</sub> #data <sub>16</sub> mem n, reg	5	F 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3	Format 31 nm 39 n:10ii 39 n:11ii 39 n:0### 37 RR ## ## 33 RR MM MM 35 RR MM MM	Bytes 2 2 2 2 4 4 4 4
	00000	men	i, iog		L L		•

#### TRAP

TRAP	Softwa	re Trap				
Syntax	TRAP	op1				
Operation	(SP) ← ((SP)) ← IF (SYS (SP) ← ((SP)) END IF (SP) ← ((SP)) ← (IP) ← 2	(SP) - 2 – (PSW 5CON.S - (SP) - ← (CSF ← 0 (SP) - 2 – (IP) zero_ex	2 GTDIS= 2 ?) 2 tend (op	0) THE 01*4)	N	
Description	Invokes operand to the s indication System entry e RETI, n execution CSP is the SG	a trap d, op1. specified on of w state xcept th return f on after pushed FDIS bit	o or inte The invo d vector thether i is prese nat the from int the trap if segm in the S	errupt oked ro table t was erved i CPU p errupt, or inte entatio SYSCO	routine utine is entry p called l dentica riority l instruc rrupt ro n is ena N regist	based on the specified determined by branching oint. This routine has no by software or hardware. Ily to hardware interrupt evel is not affected. The stion is used to resume utine has completed. The abled. This is indicated by ser.
Condition Flags	_	-		•		
	E -	-	• -	-	N -	
	E	Not a	affected		1	
	Z	Not	affected.			

Not affected.

Not affected.

Not affected.

#trap7

Format

9B t:ttt0

V

С

Ν

Mnemonic

TRAP

Addressing Modes



Bytes

XOR

XOR	Logical Exclusive OR							
Syntax	XOR	op1, op2						
Operation	(op1) ← (	op1) ⊕ (op2	)					
Data Types	WORD							
Description	Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.							
Condition Flags	Condi	tion Flags	Е	z	v	С	N	
		U	*	*	0	0	*	]
	E	Set if the van negative nu the end of a	alue of op umber. C a table.	p2 repi leared	esents otherw	the lowe ise. Use	est poss ed to sig	sible gnal
	Z	Set if result	t equals :	zero. C	leared	otherwis	e.	
	V	V Always cleared.						
	С	Always cleared.						
	Ν	Set if the Cleared oth	most się nerwise.	gnificar	nt bit of	f the re	sult is	set.
Addressing Modes	Mnemoni	С		F	ormat		Bytes	
	XOR	Rw <sub>n</sub> , Rw <sub>m</sub>		5	0 nm		2	
	XOR	Rw <sub>n</sub> , [Rw <sub>i</sub> ]		5	8 n:10ii		2	
	XOR	Rw <sub>n</sub> , [Rw <sub>i</sub> +	]	5	8 n:11ii		2	
	XOR	Rw <sub>n</sub> , #data	3	5	8 n:0##	#	2	
	XOR	reg, #data <sub>1</sub>	6	5	6 RR ##	# ##	4	
	XOR	reg, mem		5	2 RR M	IM MM	4	
	XOR	mem, reg		5	4 KK M	IM MM	4	

### XORB

XORB	Logical Exclusive OR			
Syntax	XORB op1, op2			
Operation	$(op1) \leftarrow (op1)$	op1) ⊕ (op2)		
Data Types	BYTE			
Description	Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.			
Condition Flags	_			
	E	Z V C N		
	*	* 0 0 *		
	E	Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signa the end of a table.		st possible d to signal
	Z	Set if result equals zero. Cleared otherwise.		
	V	Always cleared.		
	С	Always cleared.		
	Ν	Set if the most signification of the most signification of the set	ant bit of the res	sult is set.
Addressing Modes	Mnemonic	;	Format	Bytes
_	XORB	Rb <sub>n</sub> , Rb <sub>m</sub>	51 nm	2
	XORB	Rb <sub>n</sub> , [Rw <sub>i</sub> ]	59 n:10ii	2
	XORB	Rb <sub>n</sub> , [Rw <sub>i</sub> +]	59 n:11ii	2
	XORB	Rb <sub>n</sub> , #data <sub>3</sub>	59 n:0###	2
	XORB	reg, #data <sub>16</sub>	57 RR ## ##	4
	XORB	reg, mem	53 RR MM MM	4
	XORB	mem, reg	55 RR MM MM	4



# **SECTION 2: MAC INSTRUCTION SET**

This section describes the instruction set for the MAC co-processor. Refer to device datasheets for information about which ST10 devices include the MAC co-processor.

# 6 MAC Addressing Modes

MAC instructions use some standard ST10 addressing modes such as GPR direct or #data4 for immediate shift value. In order to supply the MAC with up to 2 new operands per instruction cycle, new MAC instruction addressing modes have been added. These allow indirect addressing with address pointer post-modification. Double indirect addressing requires 2 pointers, one of which can be supplied by any GPR, the other is provided by one of two new specific SFRs IDX0 and IDX1. Two pairs of offset registers QR0/QR1 and QX0/QX1 are associated with each pointers (GPR or IDX<sub>i</sub>). The GPR pointer allows access to the entire memory space, whereas IDX<sub>i</sub> are limited to the internal Dual-Port RAM, except for the CoMOV instruction.

The following table shows the various combinations of pointer post-modification for each of these 2 new addressing modes.

Symbol	Mnemonic	Address Pointer Operation
"[IDX <sub>i</sub> ⊗]" stands for	[IDX <sub>i</sub> ]	$(IDX_i) \leftarrow (IDX_i) \text{ (no-op)}$
	[IDX <sub>i</sub> +]	$(IDX_i) \leftarrow (IDX_i) + 2 (i=0,1)$
	[IDX <sub>i</sub> -]	$(IDX_i) \leftarrow (IDX_i) - 2 (i=0,1)$
	$[IDX_i + QX_j]$	$(IDX_i) \leftarrow (IDX_i) + (QX_j) \ (i, j = 0, 1)$
	[IDX <sub>i</sub> - QX <sub>j</sub> ]	$(IDX_i) \leftarrow (IDX_i) - (QX_j) \ (i, j = 0, 1)$
"[Rw <sub>n</sub> ⊗]" stands for	[Rwn]	$(Rwn) \leftarrow (Rwn) (no-op)$
	[Rwn+]	(Rwn) ← (Rwn) +2 (n=0-15)
	[Rwn-]	(Rwn) ← (Rwn) -2 (k=0-15)
	[Rwn + QR <sub>j</sub> ]	$(Rwn) \leftarrow (Rwn) + (QR_{j}) \ (n{=}0{-}15; j{=}0{,}1)$
	[Rwn - QR <sub>j</sub> ]	(Rwn) ← (Rwn) - (QR <sub>j</sub> ) (n=0-15; j =0,1)

### Table 6.1 Pointer post-modification for [Rwn⊗]" and "[IDXi⊗] addressing modes

In this document the symbols "[Rw\_n  $\otimes$ ]" and "[IDX\_i  $\otimes$ ]" are used to refer to these addressing modes.

133/185

### **MAC Instruction Execution Time**

A new instruction CoSTORE transfers a value from a MAC register to any location in memory. This instruction uses a specific addressing mode for the MAC registers, called **CoReg**. The following table gives the 5-bit addresses of the MAC registers corresponding to this CoReg addressing mode. Unused addresses are reserved for future revisions.

Registers	Description	Address
MSW	MAC-Unit Status Word	00000
MAH	MAC-Unit Accumulator High	00001
MAS	"limited" MAH	00010
MAL	MAC-Unit Accumulator Low	00100
MCW	MAC-Unit Control Word	00101
MRW	MAC-Unit Repeat Word	00110

 Table 6.2
 MAC register addresses for CoReg

# 7 MAC Instruction Execution Time

The instruction execution time for MAC instructions is calculated in the same way as that of the standard instruction set. To calculate the execution time for MAC instructions, refer to "Instruction Execution Times" on page 14, considering MAC instructions to be 4-byte instructions with a minimum state time number of 2.



# 8 MAC Instruction Set Summary

Mnemonic	Addressing Modes	Rep	Mnemonic	Addressing Modes	Rep
CoMUL	Rw <sub>n</sub> , Rw <sub>m</sub>	No	CoMACM	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes
CoMULu	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	CoMACMu		
CoMULus	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	CoMACMus		
CoMULsu			CoMACMsu		
CoMUL-			CoMACM-		
CoMULu-			CoMACMu-		
CoMULus-			CoMACMus-		
CoMULsu-			CoMACMsu-		
CoMUL + rnd			CoMACM + rnd		
CoMULu + rnd			CoMACMu + rnd		
CoMULus + rnd			CoMACMus + rnd		
CoMULsu + rnd			CoMACMsu + rnd		
CoMAC	Rw <sub>n</sub> , Rw <sub>m</sub>	No	CoMACMR		
CoMACu	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	CoMACMRu		
CoMACus	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	CoMACMRus		
CoMACsu			CoMACMRsu		
CoMAC-			CoMACMR + rnd		
CoMACu-			CoMACMRu + rnd		
CoMACus-			CoMACMRus + rnd		
CoMACsu-			CoMACMRsu + rnd		
CoMAC + rnd			CoADD	Rw <sub>n</sub> , Rw <sub>m</sub>	No
CoMACu + rnd			CoADD2	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes
CoMACus + rnd			CoSUB	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes
CoMACsu + rnd			CoSUB2		
CoMACR			CoSUBR		
CoMACRu			CoSUB2R		
CoMACRus			CoMAX		
CoMACRsu			CoMIN		
CoMACR + rnd			CoLOAD	Rw <sub>n</sub> , Rw <sub>m</sub>	No
CoMACRu + rnd			CoLOAD-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No
CoMACRus + rnd			CoLOAD2	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No
CoMACRsu + rnd			CoLOAD2-		
CoNOP	[Rw <sub>m</sub> ⊗]	Yes	CoCMP		
	[IDX <sub>i</sub> ⊗]	Yes	CoSHL	Rw <sub>m</sub>	Yes
	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	CoSHR	#data4	No
CoNEG	-	No	CoASHR	[Rw <sub>m</sub> ⊗]	Yes
CoNEG + rnd			CoASHR + rnd		
CoRND			CoABS	-	No
CoSTORE	Rw <sub>n</sub> , CoReg	No		Rw <sub>n</sub> , Rw <sub>m</sub>	No
	[Rw <sub>n</sub> ⊗], Core <u>g</u>	Yes			
				[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No
CoMOV	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes		Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No

### **MAC Instruction Set Summary**

The following table gives the MAC Function Code of each instruction. This Function Code is the third byte of the new instruction and is used by the co-processor as its operation code. Unused function codes are treated as CoNOP Function Code by the MAC.

Mnemonic	Function Code	Mnemonic	Function Code
CoMUL	CO	CoMACM	D8
CoMULu	00	CoMACMu	18
CoMULus	40	CoMACMus	58
CoMULsu	80	CoMACMsu	98
CoMUL-	C8	CoMACM-	E8
CoMULu-	08	CoMACMu-	28
CoMULus-	48	CoMACMus-	68
CoMULsu-	88	CoMACMsu-	A8
CoMUL + rnd	C1	CoMACM + rnd	D9
CoMULu + rnd	01	CoMACMu + rnd	19
CoMULus + rnd	41	CoMACMus + rnd	59
CoMULsu + rnd	81	CoMACMsu + rnd	99
CoMAC	D0	COMACMR	F8
CoMACu	10	CoMACMRu	38
CoMACus	50	CoMACMRus	78
CoMACsu	90	CoMACMRsu	B8
CoMAC-	E0	CoMACMR + rnd	F9
CoMACu-	20	CoMACMRu + rnd	39
CoMACus-	60	CoMACMRus + rnd	79
CoMACsu-	A0	CoMACMRsu + rnd	B9
CoMAC + rnd	D1	CoADD	02
CoMACu + rnd	11	CoADD2	42
CoMACus + rnd	51	CoSUB	0A
CoMACsu + rnd	91	CoSUB2	4A
CoMACR	F0	CoSUBR	12
CoMACRu	30	CoSUB2R	52
CoMACRus	70	СоМАХ	3A
CoMACRsu	B0	CoMIN	7A
CoMACR + rnd	F1	CoLOAD	22
CoMACRu + rnd	31	CoLOAD-	2A
CoMACRus + rnd	71	CoLOAD2	62
CoMACRsu + rnd	B1	CoLOAD2-	6A
CoNOP	5A	CoCMP	C2
CoNEG	32	CoSHL #d4	82
CoNEG + rnd	72	CoSHL other	8A
CoRND	B2	CoSHR #d4	92
CoABS -	1A	CoSHR other	9A
CoABS op1, op2	CA	CoASHR #d4	A2
CoSTORE	www:w000	CoASHR other	AA
CoMOV	00	CoASHR + rnd #d4	B2
		CoASHR + rnd other	BA

#### Table 8.2 MAC instruction function code (hexa)

# 8.1 MAC instruction conventions

This section details the conventions used to describe the MAC instruction set.

# 8.1.1 Operands

орХ	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX <sub>n</sub> )	Specifies the contents of bit n of opX
((opX))	Specifies the contents of the contents of $opX$ (i.e. $opX$ is used as pointer to the actual operand)
rnd	minus 0000 8000 <sub>H</sub>

# 8.1.2 Operations

$(opX) \leftarrow (opY)$	opY MOVED into opX
(opX) + (opY)	opY ADDED to opX
(opX) - (opY)	opY SUBTRACTED from opX
(opX) * (opY)	opY MULTIPLIED by opX
$(opX) \Leftrightarrow (opY)$	opY COMPARED against opX
(opX) <<	opX Logically SHIFTED Left
(opX) >>	opX Logically SHIFTED Right
(opX) >> <sub>a</sub>	opX Arithmetically SHIFTED Right
opX\opY	opX (MSW) and opY (LSW) CONCATENATED to a 32-bit operand.

# 8.1.3 Data addressing modes

"Rw <sub>n</sub> ", or "Rw <sub>m</sub> " :	General Purpose Registers (GPRs) where "n" and "m" are any values
r 1.	
[]:	indirect word memory location
CoReg :	MAC-Unit Register (MSW, MAH, MAL, MAS(u), MRW, MCW)
ACC :	MAC Accumulator consisting of lowest byte of MSW\MAH\MAL.
#datax :	Immediate constant (the number of significant bits is represented by 'x').

57

# 8.1.4 Instruction format

The instruction format is the same as that of the standard instruction set. In addition the following new symbols are used:

Х	4-bit IDX addressing mode encoding. (see following table)
:.qqq	3-bit GPR offset encoding for new GPR indirect with offset encoding.
rrrr:r	5-bit repeat field.
wwww:w	5-bit CoReg address for CoSTORE instructions.
SSSS:	4-bit immediate shift value.

Addressing Mode	4-bit Encoding
IDX0	1 <sub>h</sub>
IDX0 +	2 <sub>h</sub>
IDX0 -	3 <sub>h</sub>
IDX0 + QX0	4 <sub>h</sub>
IDX0 - QX0	5 <sub>h</sub>
IDX0 + QX1	6 <sub>h</sub>
IDX0 - QX1	7 <sub>h</sub>
IDX1	9 <sub>h</sub>
IDX1 +	A <sub>h</sub>
IDX1 -	B <sub>h</sub>
IDX1 + QX0	C <sub>h</sub>
IDX1 - QX0	D <sub>h</sub>
IDX1 + QX1	E <sub>h</sub>
IDX1 - QX1	F <sub>h</sub>

Table 8.3	IDX Addressing Mode Encoding and GPR offset Encoding

GPR Offset	3-bit Encoding
no-op	1 h
+	2 <sub>h</sub>
-	3 <sub>h</sub>
+ QR0	4 <sub>h</sub>
- QR0	5 <sub>h</sub>
+ QR1	6 <sub>h</sub>
- QR1	7 <sub>h</sub>

# 8.1.5 Flag states

- Unchanged
- \* Modified

# 8.1.6 Repeated Instruction Syntax

Repeatable instructions CoXXX are expressed as follows when repeated

Repeat #data5 times CoXXX ... or

### Repeat MRW times CoXXX...

When MRW is invoked, the instruction is repeated  $(MRW_{12-0}) + 1$  times, therefore the maximum number of times an instruction can be repeated is 8 192 (2<sup>13</sup>) times. In turn #data5 is an integer value that specifies the number of times an instruction is repeated, #data5 must be less than 32, and therefore in this case, CoXXX can only be repeated less than 32 times.

# 8.1.7 Shift value

The shifter authorizes only 8-bit left/right shifts. Where the specified shift value is greater than 8-bit, an 8-bit shift is performed.

# 9 Individual Instruction Description

Each instruction is described in a standard format. Refer to "MAC instruction conventions" on page 137 for detailed information about the instruction conventions.

The MAC instruction set is divided into 5 functional groups:

- Multiply and Multiply-Accumulate Instructions
- 32-Bit Arithmetic Instructions
- Shift Instructions
- Compare Instructions
- Transfer Instructions

The instructions are described within the functional groups and not in alphabetical order as for the standard instruction set.

## CoMUL(-)

CoMUL(-)	Signed Multiply Optional Round								
Group	Multiply/	Multiply-	Accumu	late Instru	uctions				
Syntax	CoMUL	c	op1, op2						
Operation	IF (MP = (ACC) ← ELSE (ACC) ← END IF	(MP = 1) THEN CC) ← ((op1) * (op2)) << 2 SE CC) ← (op1) * (op2) ID IF							
Syntax	CoMUL-		c	op1, op2					
Operation	IF (MP = (ACC) ← ELSE (ACC) ← END IF	= 1) THE ( ((op ( (op1	N 1) * (op2 ) * (op2	2)) << 2 ) ) )					
Syntax	CoMUL	c	op1, op2	, rnd					
Operation	IF (MP = 1) THEN (ACC) $\leftarrow$ ((op1) * (op2)) << 2 + 0000 8000 <sub>H</sub> ELSE (ACC) $\leftarrow$ (op1) * (op2) + 0000 8000 <sub>H</sub> END IF (MAL) $\leftarrow$ 0								
Data Types	DOUBLE	E WORD	)						
Result	32-bit si	gned val	ue						
Description	Multiplies the two signed 16-bit source operands "op1" and "op2". The obtained signed 32-bit product is first sign-extended, then and on condition MP is set, it is one-bit left shifted, and finally, it is optionally either negated or rounded before being stored in the 40-bit ACC register. The "-" option is used to negate the specified product while the "rnd" option is used to round the product using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. "rnd" and "-" are exclusive. This non-repeat-								
MAC Condition		-	•	0)/	-	0			
Flags	N *	*	0	5V -	*	5L *			
	N	Set i wise	f the mo	ost signific	ant bit of t	he result i	s set. Cleared other-		
	Z	Set i	f the res	ult equals	zero. Clea	red otherv	vise.		
	С	Alwa	iys clear	ed.					

	SV	Not affected.							
	E Always cleared when MP is cleared, otherwise, or of 8000 <sub>H</sub> by 8000 <sub>H</sub> multiplication.								
	SL	Not affected when MP or case of $8000_{\rm H}$ by $8000_{\rm H}$ m	lot affected when MP or MS are cleared, otherwise, only set in ase of $8000_{\rm H}$ by $8000_{\rm H}$ multiplication.						
Addressing Modes	Mnemonic		Rep	Format	Bytes				
	CoMUL	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm C0 00	4				
	CoMUL-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm C8 00	4				
	CoMUL	Rw <sub>n</sub> , Rw <sub>m</sub> , rnd	No	A3 nm C1 00	4				
	CoMUL	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm C0 0:0qqq	4				
	CoMUL-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm C8 0:0qqq	4				
	CoMUL	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	No	93 Xm C1 0:0qqq	4				
	CoMUL	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm C0 0:0qqq	4				
	CoMUL-	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm C8 0:0qqq	4				
	CoMUL	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd	No	83 nm C1 0:0qqq	4				
Examples	CoMUL	R0, R1, rnd	; (ACC)	← (R0)*(R1) + rnd					
	CoMUL-	R2, [R6+]	; (ACC)∢	– -(R2)*((R6))					
			; (R6) ←	· (R6) + 2					
	CoMUL	[IDX0+QX1], [R11+]	; (ACC)	← ((IDX0))*((R11))					
			; (R11)←	- (R11) + 2					
	CoMU		; (IDX0)	$\leftarrow (IDX0) + (QX1)$					
	COMOL-	[IDAT-], [KT5+QK0]	, (ACC) · (R15) ∉	$\leftarrow -((ID \times T)) ((R \times D))$ $\leftarrow (R \times D \times T) + (OR \times D)$					
			; (ICIO) ( : (IDX1)	← (IDX1) - 2					
	CoMUL	[IDX1+QX0], [R9 - QR1], rnd	; (ACC)	← ((IDX1))*((R9)) + rnd					
		-	; (R9) ←	(R9) - (QR1)					
			; (IDX1)	$\leftarrow$ (IDX1) + (QX0).					

# **Multiplication Examples**

Cases	op 1	op 2	rnd	MAE	MAH	MAL	Ν	Z	С	SV	Е	SL
MP=0, MS=x	8000 <sub>H</sub>	8000 <sub>H</sub>	0	00 <sub>H</sub>	4000 <sub>H</sub>	0000 <sub>H</sub>	0	0	0	-	0	-
MP=1, MS=0			0	00 <sub>H</sub>	8000 <sub>H</sub>	0000 <sub>H</sub>	0	0	0	-	1	-
MP=1, MS=1			0	00 <sub>H</sub>	$7FFF_{H}$	$FFFF_{H}$	0	0	0	-	0	1
MP=0, MS=x	7FFF <sub>н</sub>	$7FFF_{H}$	0	00 <sub>H</sub>	$3FFF_{H}$	0001 <sub>H</sub>	0	0	0	-	0	-
MP=1, MS=x			0	00 <sub>H</sub>	$7FFE_{H}$	0002 <sub>H</sub>	0	0	0	-	0	-
MP=1, MS=x			1	00 <sub>H</sub>	$7FFE_{H}$	0000 <sub>H</sub>	0	0	0	-	0	-
MP=0, MS=x	4001 <sub>H</sub>	F456 <sub>H</sub>	0	FF <sub>H</sub>	FD15 <sub>H</sub>	7456 <sub>н</sub>	1	0	0	-	0	-
MP=1, MS=x			0	FF <sub>H</sub>	FA2A <sub>H</sub>	E8AC <sub>H</sub>	1	0	0	-	0	-
MP=0, MS=x			1	FF <sub>H</sub>	FD15 <sub>H</sub>	0000 <sub>H</sub>	1	0	0	-	0	-
MP=1, MS=x			1	FF <sub>H</sub>	FA2B <sub>H</sub>	0000 <sub>H</sub>	1	0	0	-	0	-

## CoMULu(-)

CoMULu(-)	Unsigned Multiply Optional Round								
Group	Multiply/	Aultiply	-Accumu	late Instr	uctions				
Syntax	CoMULu		op1, op2						
Operation	$(ACC) \leftarrow$	(op1)	* (op2)						
Syntax	CoMULu	-	op1, op2						
Operation	$(ACC) \leftarrow$	- ((op1	l) * (op2)	)					
Syntax	CoMULu		op1, op2	, rnd					
Operation	$(ACC) \leftarrow$ $(MAL) \leftarrow$	(op1) 0	* (op2) +	0000 80	00 <sub>H</sub>				
Data Types	DOUBLE	WOR	D						
Result	32-bit un	signed	value						
Description	Multiply unsigned negated result is option is to round option is used, MA non-repe	the two 32-bit or rou never a used to the pr "+" and L regis atable	o unsigne product i nded bef affected k o negate roduct us d the defa ster is aut instructio	ed 16-bi s first ze fore bein by the M the spec sing two? ault round tomatica n allows	t source ro-extend g stored P mode cified pro s comple d option lly cleare up to tw	e operand ded, and flag of t duct whi ement ro is "no rou ed. "rnd" o paralle	ds "op1" and "op2". The then, it is optionally either 40-bit ACC register. The he MCW register. The "-" le the "rnd" option is used bunding. The default sign und". When "rnd" option is and "-" are exclusive. This I memory reads.		
MAC Condition	N	7	C	sv	F	91			
Flags	*	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$							
	N Set if the most significant bit of the result is set. Cleared other- wise.								
	Z	Set	if the res	ult equal	s zero. C	cleared o	therwise.		
	C	Alw	ays clear	ed.					
	SV Not affected.								

E Always cleared.

**57** 

SL Not affected.

Addressing Modes	Mnemonic		Rep	Format	Bytes
-	CoMULu	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 00 00	4
	CoMULu-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 08 00	4
	CoMULu	Rw <sub>n</sub> , Rw <sub>m</sub> , rnd	No	A3 nm 01 00	4
	CoMULu	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 00 0:0qqq	4
	CoMULu-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 08 0:0qqq	4
	CoMULu	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	No	93 Xm 01 0:0qqq	4
	CoMULu	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 00 0:0qqq	4
	CoMULu-	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 08 0:0qqq	4
	CoMULu	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd	No	83 nm 01 0:0qqq	4

The result of CoMULu is never saturated, whatever the value of MS bit is. (see multiplication examples below)

Examples	CoMULu	R0, R1, rnd	; (ACC) $\leftarrow$ (R0)*(R1) + rnd
	CoMULu-	R2, [R6+]	; (ACC) ← -(R2)*((R6))
			; (R6) ← (R6) + 2
	CoMULu	[IDX0], [R11+]	; (ACC) $\leftarrow$ ((IDX0))*((R11))
			; (R11) ← (R11) + 2
	CoMULu-	[IDX1 -], [R15+QR0]	; (ACC) ← -((IDX1))*((R15))
			; (R15) ← (R15) + (QR0)
			; (IDX1) ← (IDX1) - 2
	CoMULu	[IDX0+QX0], [R9 -], rnd	; (ACC) $\leftarrow$ ((IDX0))*((R15] + rnd
			; (R9) ← (R9) - 2
			; $(IDX0) \leftarrow (IDX0) + (QX0).$

# **Multiplication Examples**

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Ζ	С	SV	Е	SL
MP=x, MS=x	8000 <sub>H</sub>	8000 <sub>H</sub>	х	00 <sub>H</sub>	4000 <sub>H</sub>	0000 <sub>H</sub>	0	0	0	-	0	-
MP=x, MS=x	$7FFF_{H}$	$7FFF_{H}$	0	00 <sub>H</sub>	$3FFF_{H}$	0001 <sub>H</sub>	0	0	0	-	0	-
			1	00 <sub>H</sub>	3FFF <sub>н</sub>	0000 <sub>H</sub>	0	0	0	-	0	-
MP=x, MS=x	8001 <sub>H</sub>	F456 <sub>н</sub>	0	00 <sub>H</sub>	$7A2B_{H}$	F456 <sub>H</sub>	0	0	0	-	0	-
			1	00 <sub>H</sub>	7A2C <sub>H</sub>	0000 <sub>H</sub>	0	0	0	-	0	-
MP=x, MS=0	$FFFF_{H}$	$FFFF_{H}$	0	00 <sub>H</sub>	$FFFE_{H}$	0001 <sub>н</sub>	0	0	0	-	0	-
MP=x, MS=1			0	00 <sub>H</sub>	7FFF <sub>H</sub>	$FFFF_{H}$	0	0	0	-	0	-

## CoMULsu(-)

CoMULsu(-)	Mixed Multiply Optional Round								
Group	Multiply/M	Multiply/Multiply-Accumulate Instructions							
Syntax	CoMULsu	C	op1, op2						
Operation	$(ACC) \leftarrow ($	op1) *	(op2)						
Syntax	CoMULsu-	c	op1, op2						
Operation	$(ACC) \leftarrow -$	((op1)	) * (op2))						
Syntax	CoMULsu	C	op1, op2,	rnd					
Operation	$(ACC) \leftarrow (op1) * (op2) + 0000 8000_H$ $(MAL) \leftarrow 0$								
Data Types		VORD	)						
Result	32-bit sign	ed valu	ue						
Description	Multiply the two 16-bit signed and unsigned source operands "op1" and "op2", respectively. The obtained signed 32-bit product is first sign-extended, then, it is optionally either negated or rounded before being stored in the 40-bit ACC register. The result is never affected by the MP mode flag contained in the MCW register. The "-" option is used to negate the specified product while the "rnd" option is used to round the product using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. "rnd" and "-" are exclusive. This non-repeatable instruction allows up to two								
MAC Condition	N	7	C	<u>ev</u>	F	ei			
Flags	N *	*	0	-	0	- -			
	N	Set i wise	f the mos	st signific	cant bit	of the res	sult is set. Cleared other-		
	Z	Z Set if the result equals zero. Cleared otherwise.							

- C Always cleared.
- SV Not affected.
- E Always cleared.
- SL Not affected.
| Addressing Modes | Mnemonic |  | Rep      | Format                              | Bytes |
|------------------|----------|--|----------|-------------------------------------|-------|
| -                | CoMULsu  | Rw <sub>n</sub> , Rw <sub>m</sub>              | No       | A3 nm 80 00                         | 4     |
|                  | CoMULsu- | Rw <sub>n</sub> , Rw <sub>m</sub>              | No       | A3 nm 88 00                         | 4     |
|                  | CoMULsu  | Rw <sub>n</sub> , Rw <sub>m</sub> , rnd        | No       | A3 nm 81 00                         | 4     |
|                  | CoMULsu  | [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]      | No       | 93 Xm 80 0:0qqq                     | 4     |
|                  | CoMULsu- | [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]      | No       | 93 Xm 88 0:0qqq                     | 4     |
|                  | CoMULsu  | [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd | No       | 93 Xm 81 0:0qqq                     | 4     |
|                  | CoMULsu  | Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]          | No       | 83 nm 80 0:0qqq                     | 4     |
|                  | CoMULsu- | Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]          | No       | 83 nm 88 0:0qqq                     | 4     |
|                  | CoMULsu  | Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd     | No       | 83 nm 81 0:0qqq                     | 4     |
| Examples         | CoMULsu  | R0, R1, rnd                                    | ; (ACC)  | ← (R0)*(R1) + rnd                   |       |
| •                | CoMULsu- | R2, [R6+]                                      | ; (ACC)  | ← -(R2)*((R6))                      |       |
|                  |          |  | ; (R6) ← | – (R6) + 2                          |       |
|                  | CoMULsu  | [IDX0], [R11+]                                 | ; (ACC)  | $\leftarrow$ ((IDX0))*((R11))       |       |
|                  |          |  | ; (R11)  | ← (R11) + 2                         |       |
|                  | CoMULsu- | [IDX1 -], [R15]                                | ; (ACC)  | ← -((IDX1))*((R15))                 |       |
|                  |          |  | ; (IDX1) | ) ← (IDX1) - 2                      |       |
|                  | CoMULsu  | [IDX0+QX0], [R9 - QR1], rnd                    | ; (ACC)  | $\leftarrow ((IDX0))^*((R9)) + rnd$ |       |
|                  |          |  | ; (R9) ← | – (K9) - (QR1)                      |       |
|                  |          |  | ; (IDX0) | $i \leftarrow (IDX0) + (QX0).$      |       |

# **Multiplication Examples**

Cases	op 1	op 2	rnd	MAE	MAH	MAL	Ν	Z	С	sv	Е	SL
MP=x, MS=x	8000 <sub>H</sub>	8000 <sub>H</sub>	х	FF <sub>H</sub>	C000 <sub>H</sub>	0000 <sub>H</sub>	1	0	0	-	0	-
MP=x,	7FFF <sub>H</sub>	$7FFF_{H}$	0	00 <sub>H</sub>	3FFF <sub>H</sub>	0001 <sub>H</sub>	0	0	0	-	0	-
MS=x			1	00 <sub>H</sub>	3FFF <sub>H</sub>	0000 <sub>H</sub>	0	0	0	-	0	-
MP=x,	8001 <sub>H</sub>	F456 <sub>н</sub>	0	FF <sub>H</sub>	85D5 <sub>н</sub>	F456 <sub>н</sub>	1	0	0	-	0	-
MS=x			1	FF <sub>H</sub>	85D6 <sub>H</sub>	0000 <sub>H</sub>	1	0	0	-	0	-

#### CoMULus(-)

CoMULus(-)	Mixed Multiply Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMULus op1, op2
Operation	$(ACC) \leftarrow (op1) * (op2)$
Syntax	CoMULus- op1, op2
Operation	(ACC) ← - ((op1) * (op2))
Syntax	CoMULus op1, op2, rnd
Operation	$(ACC) \leftarrow (op1) * (op2) + 0000 8000_H$ (MAL) $\leftarrow 0$
Data Types	DOUBLE WORD
Result	32-bit signed value
Description	Multiply the two 16-bit unsigned and signed "op2", respectively. The obtained signed 32-bit then it is optionally either negated or rounded bit bit ACC register. The result is power affe

Multiply the two 16-bit unsigned and signed source operands "op1" and "op2", respectively. The obtained signed 32-bit product is first sign-extended, then it is optionally either negated or rounded before being stored in the 40-bit ACC register. The result is never affected by the MP mode flag contained in the MCW register. The "-" option is used to negate the specified product while the "rnd" option is used to round the product using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. "rnd" and "-" are exclusive. This non-repeatable instruction allows up to two parallel memory reads.

## MAC Condition Flags

Ν	Z	С	SV	Е	SL	
*	*	0	-	0	-	]
N	Set i wise.	f the mo	st signifi	cant bit d	of the re	sult is set. Cleared other-

- Z Set if the result equals zero. Cleared otherwise.
- C Always cleared.
- SV Not affected.
- E Always cleared.
- SL Not affected.

Addressing Modes	Mnemonic		Rep	Format	Bytes
-	CoMULus	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 40 00	4
	CoMULus-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 48 00	4
	CoMULus	Rw <sub>n</sub> , Rw <sub>m</sub> , rnd	No	A3 nm 41 00	4
	CoMULus	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 40 0:0qqq	4
	CoMULus-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 48 0:0qqq	4
	CoMULus	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	No	93 Xm 41 0:0qqq	4
	CoMULus	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 40 0:0qqq	4
	CoMULus-	Rwn, [Rw <sub>m</sub> ⊗]	No	83 nm 48 0:0qqq	4
	CoMULus	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd	No	83 nm 41 0:0qqq	4
Examples	CoMULus	R0, R1, rnd	; (ACC)	$\leftarrow$ (R0)*(R1) + rnd	
•	CoMULus-	R2, [R6+]	; (ACC)	← -(R2)*((R6))	
			; (R6) ←	- (R6) + 2	
	CoMULus	[IDX1+QX0], [R11+QR0]	; (ACC)	← ((IDX1))*((R11))	
			; (R11)	← (R11) + (QR0)	
			; (IDX1)	$\leftarrow$ (IDX1) + (QX0)	
	CoMULus-	[IDX0], [R15]	; (ACC)	← -((IDX0))*((R15))	
	CoMULus	[IDX0+QX0], [R9 - QR1], rnd	; (ACC)	$\leftarrow ((IDX0))^*((R15] + rnd)$	
			; (R9) ←	- (R9) - (QR1)	
			; (IDX0)	$\leftarrow (IDX0) + (QX0).$	

# **Multiplication Examples**

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Z	С	SV	Е	SL
MP=x, MS=x	8000 <sub>H</sub>	8000 <sub>H</sub>	х	FF <sub>H</sub>	C000 <sub>H</sub>	0000 <sub>H</sub>	1	0	0	-	0	-
MP=x, MS=x	$7FFF_{H}$	$7FFF_{H}$	0	00 <sub>H</sub>	$3FFF_{H}$	0001 <sub>H</sub>	0	0	0	-	0	-
			1	00 <sub>H</sub>	$3FFF_{H}$	0000 <sub>H</sub>	0	0	0	-	0	-
MP=x, MS=x	8001 <sub>H</sub>	F456 <sub>H</sub>	0	FF <sub>H</sub>	FA2A <sub>H</sub>	0001 <sub>H</sub>	1	0	0	-	0	-
			1	FF <sub>H</sub>	$FA2B_{H}$	0000 <sub>H</sub>	1	0	0	-	0	-

# CoMAC(R/-)

CoMAC(R/-)	Multiply-Accumulate Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMAC op1, op2
Operation	IF (MP = 1) THEN (tmp) $\leftarrow$ ((op1) * (op2)) << 2 (ACC) $\leftarrow$ (ACC) + (tmp) ELSE (tmp) $\leftarrow$ (op1) * (op2) (ACC) $\leftarrow$ (ACC) + (tmp) END IF
Syntax	CoMAC op1, op2, rnd
Operation	$\begin{split} & IF\;(MP=1)\;THEN\\ & (tmp) \leftarrow ((op1)*(op2)) << 2\\ & (ACC) \leftarrow (ACC) + (tmp) + 0000\;8000_{H}\\ & ELSE\\ & (tmp) \leftarrow (op1)*(op2)\\ & (ACC) \leftarrow (ACC) + (tmp) + 0000\;8000_{H}\\ & END\;IF\\ & (MAL) \leftarrow 0 \end{split}$
Syntax	CoMAC- op1, op2
Operation	$IF (MP = 1) THEN$ $(tmp) \leftarrow ((op1) * (op2)) << 2$ $(ACC) \leftarrow (ACC) - (tmp)$ $ELSE$ $(tmp) \leftarrow (op1) * (op2)$ $(ACC) \leftarrow (ACC) - (tmp)$ $END IF$
Syntax	CoMACR op1, op2
Operation	IF (MP = 1) THEN (tmp) $\leftarrow$ ((op1) * (op2)) << 2 (ACC) $\leftarrow$ (tmp) - (ACC) ELSE (tmp) $\leftarrow$ (op1) * (op2) (ACC) $\leftarrow$ (tmp) - (ACC) END IF



Syntax	CoMACR op1, op2, rnd
Operation	$\begin{split} & IF \; (MP=1) \; THEN \\ & (tmp) \leftarrow ((op1) * (op2)) << 2 + 0000 \; 8000_{H} \\ & (ACC) \leftarrow (tmp) - (ACC) \\ & ELSE \\ & (tmp) \leftarrow (op1) * (op2) \\ & (ACC) \leftarrow (tmp) - (ACC) + 0000 \; 8000_{H} \\ & END \; IF \\ & (MAL) \leftarrow 0 \end{split}$
Data Types	DOUBLE WORD
Result	40-bit signed value
Description	Multiplies the two signed 16-bit source operands "op1" and "op2". The obtained signed 32-bit product is first sign-extended, then and on condition MP flag is set, it is one-bit left shifted, next, it is optionally negated prior being added/subtracted to/from the 40-bit ACC register content, finally, the obtained result is optionally rounded before being stored in the 40-bit ACC register. "-" option is used to negate the specified product, "R" option is used

stored in the 40-bit ACC product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and allows up to two parallel memory reads.

## **MAC Condition** Flags

57

Ν	Z	С	SV	Е	SL	
*	*	*	*	*	*	]
N	Set if wise.	the mo	st signifi	cant bit o	of the re	sult is set. Cleared othe
Z	Set if	the res	ult equals	s zero. C	leared o	therwise.
С	Set if	a carry	or borrow	w is gene	erated. C	cleared otherwise.
SV	Set if	an arith	metic ov	erflow o	curred.	Not affected otherwise.
E	Set if	the MA	E is usec	I. Cleare	d otherw	vise.
SL	Set in affect	f the co ted othe	ntents of rwise.	the AC	C is aut	tomatically saturated. No

149/185

# CoMAC(R/-)

Addressing Modes	Mnemonic		Rep	Format	Bytes			
	CoMAC	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm D0 00	4			
	CoMAC-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm E0 00	4			
	CoMAC	Rw <sub>n</sub> , Rw <sub>m</sub> , rnd	No	A3 nm D1 00	4			
	CoMACR	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm F0 00	4			
	CoMACR	Rw <sub>n</sub> , Rw <sub>m</sub> , rnd	No	A3 nm F1 00	4			
	CoMAC	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm D0 rrrr:rqqq	4			
	CoMAC-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm E0 rrrr:rqqq	4			
	CoMAC	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	Yes	93 Xm D1 rrrr:rqqq	4			
	CoMACR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm F0 rrrr:rqqq	4			
	CoMACR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	Yes	93 Xm F1 rrrr:rqqq	4			
	CoMAC	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm D0 rrrr:rqqq	4			
	CoMAC-	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm E0 rrrr:rqqq	4			
	CoMAC	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd	Yes	83 nm D1rrrr:rqqq	4			
	CoMACR	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm F0 rrrr:rqqq	4			
	CoMACR	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗], rnd	Yes	83 nm F1 rrrr:rqqq	4			
Examples	CoMAC	R3, R4, rnd	; (ACC	C) ← (ACC) + (R3)*(R4) +	- rnd			
	CoMAC-	R2, [R6+]	; (ACC	C) ← (ACC) - (R2)*((R6))				
			; (R6)	← (R6) + 2				
	CoMAC	[IDX0+QX0], [R11+QR0]	; (ACC	$C) \leftarrow (ACC) + ((IDX0))^*((R)^*)$	11))			
			; (R11	$) \leftarrow (R11) + (QR0)$				
	; $(IDX0) \leftarrow (IDX0) + (QX0)$							
	Repeat 3 tin	ies coniac [idx1 - QX1], [Re	+QR [] · (ΔC(	$C \leftarrow (ACC) + ((IDX1))*((RS))$	a))			
			; (/.ec	$\leftarrow (R9) + (QR1)$	5))			
			; (IDX	1) ← (IDX1) - (QX1)				
	Repeat MR\	N times CoMAC - R3, [R7 - Q	R0] ; (ACC	C) ← (ACC) - (R3)*((R7))				
			; (R7)	← (R7) - (QR0)				
	CoMACR	[IDX1], [R4+], rnd	; ACC ; (R4)	) ← ((IDX1))*((R4)) - (Acc) ← (R4) + 2	+ rnd			



# **CoMAC(R)u(-)** Unsigned Multiply-Accumulate Optional Round

Group	Multiply/Multiply	y-Accumulate Instructions
Syntax	CoMACu	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (ACC)^{\frac{1}{2}}$	∗ (op2) ) + (tmp)
Syntax	CoMACu	op1, op2, rnd
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (ACC)$ $(MAL) \leftarrow 0$	<sup></sup> ∗ (op2) ) + (tmp) + 0000 8000 <sub>H</sub>
Syntax	CoMACu-	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ (ACC) $\leftarrow (ACC)^{\frac{1}{2}}$	∗ (op2) ) - (tmp)
Syntax	CoMACRu	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (tmp)^{\frac{1}{2}}$	* (op2) - (ACC)
Syntax	CoMACRu	op1, op2, rnd
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (tmp)$ $(MAL) \leftarrow 0$	∗ (op2) - (ACC) + 0000 8000 <sub>H</sub>
Data Types	DOUBLE WOR	D
Result	40-bit unsigned	l value
Description	Multiplies the t obtained unsign negated prior content, finally, in the 40-bit AC contained in the product, "R" op "rnd" option is The default sig When "rnd" opt "rnd" and "-" ar repeated and a	wo unsigned 16-bit source operands "op1" and "op2". The ned 32-bit product is first zero-extended and then optionally being added/subtracted to/from the 40-bit ACC register the obtained result is optionally rounded before being stored C register. The result is never affected by the MP mode flag the MCW register. "-" option is used to negate the specified botion is used to negate the accumulator content, and finally used to round the result using two's complement rounding. In option is "+" and the default round option is "no round". tion is used, MAL register is automatically cleared. Note that re exclusive as well as "-" and "R". This instruction might be llows up to two parallel memory reads.

MAC Condition											
Flags	N	Z	С	SV	E	SL	_				
-	*	*	*	*	*	*					
	Ν	N Set if the most significant bit of the result is set. Cleared other- wise.									
	Z	Set if the result equals zero. Cleared otherwise.									
	С	Set i	f a carry	or borrow	w is ger	nerated. C	Cleared otherwise.				
	SV	Set i	f an arith	metic ov	erflow c	occurred.	Not affected othe	rwise.			
	Е	Set i	f the MA	E is used	l. Clear	ed otherv	vise.				
	SL	Set affec	if the co ted othe	ntents of rwise.	the A	CC is au	tomatically satura	ted. Not			
Addressing Modes	Mnemonic				Re	p Fo	ormat	Bytes			
	CoMACu	Rv	v <sub>n</sub> , Rw <sub>m</sub>		No	A:	3 nm 10 00	4			
	CoMACu-	Rv	v <sub>n</sub> , Rw <sub>m</sub>		No	A AS	3 nm 20 00	4			
	CoMACu	Rv	v <sub>n</sub> , Rw <sub>m</sub> ,	rnd	No	A3	3 nm 11 00	4			
	CoMACRu	Rv	v <sub>n</sub> , Rw <sub>m</sub>		No	A:	3 nm 30 00	4			
	CoMACRu	Rv	v <sub>n</sub> , Rw <sub>m</sub> ,	rnd	No	A	3 nm 31 00	4			
	CoMACu	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]			Ye	s 93	3 Xm 10 rrrr:rqqq	4			
	CoMACu-	[ID	X <sub>i</sub> ⊗], [R	 ∧m⊗]	Ye	s 93	3 Xm 20 rrrr:rqqq	4			
	CoMACu	[ID	X <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗], rn	d Ye	s 93	3 Xm 11 rrrr:rqqq	4			
	CoMACRu	[ID	X <sub>i</sub> ⊗], [R	 N <sub>m</sub> ⊗]	Ye	s 93	3 Xm 30 rrrr:rqqq	4			
	CoMACRu	- [ID		w <sub>m</sub> ⊗], rn	d Ye	s 93	3 Xm 31 rrrr:rqqq	4			
	CoMACu	Rv	v <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Ye	s 83	nm 10 rrrr:rqqq	4			
	CoMACu-	Rv	v <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Ye	s 83	3 nm 20 rrrr:rqqq	4			
	CoMACu	Rv	v <sub>n</sub> , [Rw <sub>m</sub>	⊗], rnd	Ye	s 83	nm 11 rrrr:rqqq	4			
	CoMACRu	Rv	v <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Ye	s 83	3 nm 30 rrrr:rggg	4			
	CoMACRu	Rv	v <sub>n</sub> , [Rw <sub>m</sub>	⊗], rnd	Ye	s 83	3 nm 31 rrrr:rqqq	4			
Examples	CoMACu	R5,	R8, rnd		:	(ACC) ← (	ACC) + (R5)*(R8) +	rnd			
	CoMACu-	R2,	[R7]		;	$(ACC) \leftarrow ($	ACC) - (R2)*((R7))				
	CoMACu	[IDX(	) - QX0], [	R11 - QR(	)];	$(ACC) \leftarrow ($	ACC) + ((IDX0))*((R	11))			
					;	(R11) ← (l	R11) - (QR0)				
					;	$(IDX0) \leftarrow$	(IDX0) - (QX0)				
	Repeat 3 tim	nes Co	MACu [ID	X1+], [R9	-] ;	$(ACC) \leftarrow ($	ACC) + ((IDX1))*((R	9))			
					;	(R9) ← (R	9) - 2				
	; $(IDX1) \leftarrow (IDX1) + 2$										
	Repeat WRV	v times		и- кз, [К1	- QKUJ;	$(AUU) \leftarrow ($	(R, C,				
	CoMACRU	זאטוז		R41 rnd	,	$(ACC) \leftarrow (R$	//DX1))*((R4))-(ACC	:)+ rnd			
		1.57	. <u></u> ,	],a	;	; $(IDX1) \leftarrow (IDX1) - (QX0)$					

CoMAC(R)su(-)	Mixed Multiply-Accumulate Optional Round
---------------	--

Group	Multiply/Multiply	y-Accumulate Instructions
Syntax	CoMACsu	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ (ACC) $\leftarrow (ACC)^{\frac{1}{2}}$	∗ (op2) ) + (tmp)
Syntax	CoMACsu	op1, op2, rnd
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (ACC)$ $(MAL) \leftarrow 0$	<sup></sup> ∗ (op2) ) + (tmp) + 0000 8000 <sub>H</sub>
Syntax	CoMACsu-	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ (ACC) $\leftarrow (ACC)^{\frac{1}{2}}$	∗ (op2) ) - (tmp)
Syntax	CoMACRsu	op1, op2
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (tmp)^{\frac{1}{2}}$	∗ (op2) - (ACC)
Syntax	CoMACRsu	op1, op2, rnd
Operation	$(tmp) \leftarrow (op1)^{\frac{1}{2}}$ $(ACC) \leftarrow (tmp)$ $(MAL) \leftarrow 0$	<sup>∗</sup> (op2) - (ACC) + 0000 8000 <sub>H</sub>
Data Types	DOUBLE WOR	D
Result	40-bit signed va	alue
Description	Multiplies the t "op2", respective and then, it is 40-bit ACC reg before being star the MP mode fl the specified pri and finally "rnd rounding. The round". When Note that "rnd" might be repea	wo signed and unsigned 16-bit source operands "op1" and vely. The obtained signed 32-bit product is first sign-extended, optionally negated prior being added/subtracted to/from the ister content, finally the obtained result is optionally rounded ored in the 40-bit ACC register. The result is never affected by ag contained in the MCW register. "-" option is used to negate roduct, "R" option is used to negate the accumulator content, " option is used to round the result using two's complement default sign option is "+" and the default round option is "no "rnd" option is used, MAL register is automatically cleared. and "-" are exclusive as well as "-" and "R". This instruction ted and allows up to two parallel memory reads.

MAC Condition								
Flags	Ν	Z	С	SV	E	SL	-	
	*	*	*	*	*	*		
	Ν	Set wise	if the mo	ost signifio	cant bit o	of the re	sult is set. Cleare	d other-
	Z	Set	if the res	ult equals	s zero. C	leared o	therwise.	
	С	Set	if a carry	or borrow	w is gene	erated. C	leared otherwise.	
	SV	Set	if an arith	nmetic ov	erflow od	curred.	Not affected other	wise.
	E	Set	if the MA	E is used	l. Cleare	d otherw	vise.	
	SL	Set affe	if the co cted othe	ontents of erwise.	the AC	C is aut	omatically satura	ted. Not
Addressing Modes	Mnemonic				Rep	o Fo	rmat	Bytes
	CoMACsu	R	w <sub>n</sub> , Rw <sub>m</sub>		No	A3	8 nm 90 00	4
	CoMACsu-	R	w <sub>n</sub> , Rw <sub>m</sub>		No	A3	8 nm A0 00	4
	CoMACsu	R	w <sub>n</sub> , Rw <sub>m</sub> ,	rnd	No	A3	8 nm 91 00	4
	CoMACRsu	ı R	w <sub>n</sub> , Rw <sub>m</sub>		No	A3	8 nm B0 00	4
	CoMACRsu	ı R	w <sub>n</sub> , Rw <sub>m</sub> ,	rnd	No	A3	8 nm B1 00	4
	CoMACsu	[[[	OX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm 90 rrrr:rqqq	4
	CoMACsu-	[[[	ΟΧ <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm A0 rrrr:rqqq	4
	CoMACsu	[[[	OX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗], rn	d Yes	93	Xm 91 rrrr:rqqq	4
	CoMACRsu	J [ [	OX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm B0 rrrr:rqqq	4
	CoMACRsu	J [ [	OX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗], rn	d Yes	93	Xm B1 rrrr:rqqq	4
	CoMACsu	R	w <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Yes	83	nm 90 rrrr:rqqq	4
	CoMACsu-	R	w <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Yes	83	nm A0 rrrr:rqqq	4
	CoMACsu	R	w <sub>n</sub> , [Rw <sub>m</sub>	⊗], rnd	Yes	83	nm 91 rrrr:rqqq	4
	CoMACRsu	ı R	w <sub>n</sub> , [Rw <sub>m</sub>	⊗]	Yes	83	nm B0 rrrr:rqqq	4
	CoMACRsu	א R	w <sub>n</sub> , [Rw <sub>m</sub>	⊗], <b>rnd</b>	Yes	83	nm B1 rrrr:rqqq	4
Examples	CoMACsu CoMACsu-	R5, R2,	R8, rnd [R7]		; (/ ; (/	() → (COA () → (COA	ACC) + (R5)*(R8) + ACC) - (R2)*((R7))	rnd
	CoMACsu	[IDX	0 - QX0], [	R11 - QR0	)] ; ( <i>i</i>	$ACC) \leftarrow ($	ACC) + ((IDX0))*((R	11))
					; (I	R11) ← (F	R11) - (QR0)	
					; (I	DX0) ← (	IDX0) - (QX0)	
	Repeat 3 tim	es Co	MACsu [II	DX1+], [R9	9-] ; (/	$ACC) \leftarrow (ACC)$	ACC) + ((IDX1))*((R	9))
					; (I	R9) ← (R9	9) - 2	
		/ time		D2 [D]	; (I 7 OP01	DX1) ← (	IDX1) + 2	
		, une		su -ns, [N	· - QRUj · (.	$ACC) \leftarrow ($	ACC) - (R3)*((R7))	
					; (/	.30,	7) - (QR0)	
	CoMACsur	[IDX	1 - QX0], [	R4], rnd	; (/	, (OOA	(IDX1))*((R4)) - (AC	C)
			-		; (I	DX1) ← (	IDX1) - (QX0)	

# CoMAC(R)us(-) Mixed Multiply-Accumulate Optional Round

Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMACus op1, op2
Operation	$(tmp) \leftarrow (op1) * (op2)$ $(ACC) \leftarrow (ACC) + (tmp)$
Syntax	CoMACus op1, op2, rnd
Operation	(tmp) ← (op1) * (op2) (ACC) ← (ACC) + (tmp) + 0000 8000 <sub>H</sub> (MAL) ← 0
Syntax	CoMACus- op1, op2
Operation	$(tmp) \leftarrow (op1) * (op2)$ $(ACC) \leftarrow (ACC) - (tmp)$
Syntax	CoMACRus op1, op2
Operation	$(tmp) \leftarrow (op1) * (op2)$ $(ACC) \leftarrow (tmp) - (ACC)$
Syntax	CoMACRus op1, op2, rnd
Operation	$(tmp) \leftarrow (op1) * (op2)$ (ACC) $\leftarrow (tmp) - (ACC) + 0000 8000_H$ (MAL) $\leftarrow 0$
Data Types	DOUBLE WORD
Result	40-bit signed value
Description	Multiplies the two unsigned and signed 16-bit source operands "op1" and "op2", respectively. The obtained signed 32-bit product is first sign-extended, and then, it is optionally negated prior being added/subtracted to/from the 40-bit ACC register content, finally the obtained result is optionally rounded before being stored in the 40-bit ACC register. The result is never affected by the MP mode flag contained in the MCW register. "-" option is used to negate the specified product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and allows up to two parallel memory reads.

MAC Condition								
Flags	Ν	Ζ	С	SV	Е	SL	_	
U	*	*	*	*	*	*		
	N	Set wis	t if the mo e.	ost signifi	cant bit o	of the re	sult is set. Cleare	ed other-
	Z	Set	if the res	ult equals	s zero. C	leared of	therwise.	
	С	Set	if a carry	or borrov	v is gene	erated. C	leared otherwise.	
	SV	Set	if an arith	nmetic ov	erflow od	curred.	Not affected othe	rwise.
	Е	Set	t if the MA	E is used	I. Cleare	d otherw	ise.	
	SL	Set affe	t if the co	ontents of erwise.	the AC	C is aut	omatically satura	ted. Not
Addressing Modes	Mnemoni	с			Rep	o Fo	rmat	Bytes
	CoMACu	s R	Rw <sub>n</sub> , Rw <sub>m</sub>		No	A3	nm 50 00	4
	CoMACu	s- R	Rw <sub>n</sub> , Rw <sub>m</sub>		No	A3	nm 60 00	4
	CoMACu	s R	Rw <sub>n</sub> , Rw <sub>m</sub> ,	, rnd	No	A3	nm 51 00	4
	CoMACR	us R	Rw <sub>n</sub> , Rw <sub>m</sub>		No	A3	nm 70 00	4
	CoMACR	us R	Rw <sub>n</sub> , Rw <sub>m</sub>	, rnd	No	A3	nm 71 00	4
	CoMACu	s [l	DX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm 50 rrrr:rqqq	4
	CoMACu	s- [l	DX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm 60 rrrr:rqqq	4
	CoMACu	s [l	DX <sub>i</sub> ⊗], [R	.w <sub>m</sub> ⊗], rno	d Yes	93	Xm 51 rrrr:rqqq	4
	CoMACR	us [l	DX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗]	Yes	93	Xm 70 rrrr:rqqq	4
	CoMACR	us [l	DX <sub>i</sub> ⊗], [R	w <sub>m</sub> ⊗], rno	d Yes	93	Xm 71 rrrr:rqqq	4
	CoMACu	s R	Rw <sub>n</sub> , [Rw <sub>m</sub>	,⊗]	Yes	83	nm 50 rrrr:rqqq	4
	CoMACu	s- R	Rw <sub>n</sub> , [Rw <sub>m</sub>	∖⊗]	Yes	83	nm 60 rrrr:rqqq	4
	CoMACu	s R	Rw <sub>n</sub> , [Rw <sub>m</sub>		Yes	83	nm 61 rrrr:rqqq	4
	CoMACR	us R	Rw <sub>n</sub> , [Rw <sub>m</sub>	 ∖⊗]	Yes	83	nm 70 rrrr:rqqq	4
	CoMACR	us R	Rw <sub>n</sub> , [Rw <sub>m</sub>	<sub>n</sub> ⊗], rnd	Yes	83	nm 71 rrrr:rqqq	4
Examples	CoMACus	R5	, R8, rnd		; (/	$ACC) \leftarrow (\lambda$	ACC) + (R5)*(R8) +	rnd
	CoMACus-	• R2	, [R7]		; (/	$ACC) \leftarrow (ACC)$	ACC) - (R2)*((R7))	
	CoMACus	[ID>	<0 - QX0],	[R11 - QR0	)] ; (/	$ACC) \leftarrow (\lambda$	ACC) + ((IDX0))*((R	11))
					; (I	R11) ← (F	R11) - (QR0)	
					; (	$ DX0\rangle \leftarrow ($	IDX0) - (QX0)	0))
	Repeat 3 t	imes C	omacus[IL	JX1+J, [R9	-];(/	$ACC) \leftarrow (R$	ACC) + ((IDX1))^((R	9))
					; (1	K9) ← (K: IDY1) ∠ (	9) - 2 IDX1) + 2	
	Repeat MF	RW time	es CoMAC	us-R3 [R7	, (i - QR0] <sup>.</sup> (	$D \land I ) \leftarrow ($ ACC) $\leftarrow ($	ACC) - (R3)*((R7))	
	. top out MI				<u>ح</u> . رە], ( : (ا	R7) ← (R7	7) - (QR0)	
	CoMACRu	s [ID>	(1 - QX0],	[R4], rnd	;(A	, ∧CC) ← ((	IDX1))*((R4))-(ACC)	)+rnd
					; (I	DX1) ← (	IDX1) - (QX0)	

CoMACM(R/-)	Multiply-Accumulate Parallel Data Move and Optional Round				
Group	Multiply/Multiply	y-Accumulate Inst	ructions		
Syntax	CoMACM	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]			
Operation	$\begin{array}{l} IF (MP=1) \ TH \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC) \\ ELSE \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC) \\ END \ IF \end{array}$	EN ))*((Rw <sub>m</sub> )) << 2 ;) + (tmp) ))*((Rw <sub>m</sub> )) ;) + (tmp)	$((IDX_i(-\otimes))) \gets ((IDX_i))$		
Syntax	CoMACM	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗],	rnd		
Operation	$\begin{array}{l} IF (MP=1) \ TH \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC) \\ ELSE \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC) \\ END \ IF \\ (MAL) \leftarrow 0 \end{array}$	EN )*((Rw <sub>m</sub> )) << 2 2) + (tmp) + 0000 8 1)*((Rw <sub>m</sub> )) 2) + (tmp) + 0000 8	$0000_{H}$ $0000_{H}$ $((IDX_{i}(-\otimes))) \leftarrow ((IDX_{i}))$		
Syntax	CoMACM-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]			
Operation	$\begin{array}{l} IF (MP=1) \ TH \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC \\ ELSE \\ (tmp) \leftarrow ((IDX_i) \\ (ACC) \leftarrow (ACC \\ END \ IF \end{array}$	EN ))*((Rw <sub>m</sub> )) << 2 ;) + (tmp) ))*((Rw <sub>m</sub> )) ;) - (tmp)	$((IDX_i(-\otimes))) \gets ((IDX_i))$		
Syntax	CoMACMR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]			
Operation	$IF (MP = 1) TH (tmp) \leftarrow ((IDX_i) (ACC) \leftarrow (-ACC ELSE (tmp) \leftarrow ((IDX_i) (ACC) \leftarrow (-ACC END IF $	EN ))*((Rw <sub>m</sub> )) << 2 C) + (tmp) ))*((Rw <sub>m</sub> )) C) + (tmp)	$((IDX_{i}(-\otimes))) \leftarrow ((IDX_{i}))$		
Syntax	CoMACMR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗],	rnd		
Operation	$\begin{array}{l} IF \; (MP = 1) \; TH \\ (tmp) \leftarrow ((IDX_{i}) \end{array}$	EN )*((Rw <sub>m</sub> )) << 2			

	$(ACC) \leftarrow (-ACC) + (tmp)$	b) + 0000 8000 <sub>H</sub>
	ELSE	
	$(tmp) \leftarrow ((IDX_{i}))^*((Rw_{m}))$	
	$(ACC) \leftarrow (-ACC) + (tmp)$	) + 0000 8000 <sub>н</sub>
	END IF	
	$(MAL) \leftarrow 0$	$((IDX_{i}(-\otimes))) \leftarrow ((IDX_{i}))$
Data Types	DOUBLE WORD	
Result	40-bit signed value	
Description	Multiplies the two signer and Rw <sub>m</sub> (m=0-15), res sign-extended, then an shifted, and next, it is from the 40-bit ACC res rounded before being s	ed 16-bit source operands pointed to by $IDX_i$ (i=0,1) spectively. The obtained signed 32-bit product is first ad on condition the MP flag is set, it is one-bit left optionally negated prior being added/subtracted to/ gister content, finally the obtained result is optionally tored in the 40-bit ACC register. "-" option is used to

shifted, and next, it is optionally negated prior being added/subtracted to/ from the 40-bit ACC register content, finally the obtained result is optionally rounded before being stored in the 40-bit ACC register. "-" option is used to negate the specified product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and performs two parallel memory reads. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDX<sub>i</sub> overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX<sub>i</sub>, as explained by the following table

Addressing Mode	Overwritten Address
[IDX <sub>i</sub> ]	(no change)
[IDX <sub>i</sub> +]	(IDX <sub>i</sub> ) - 2
[IDX <sub>i</sub> -]	(IDX <sub>i</sub> ) + 2
[IDX <sub>i</sub> +QX <sub>j</sub> ]	(IDX <sub>i</sub> ) - (QX <sub>j</sub> )
[IDX <sub>i</sub> -QXj]	$(IDX_i) + (QX_j)$

### MAC Condition Flags

Ν	Z	С	SV	Е	SL
*	*	*	*	*	*

- N Set if the most significant bit of the result is set. Cleared otherwise.
- Z Set if the result equals zero. Cleared otherwise.
- C Set if a carry or borrow is generated. Cleared otherwise.
- SV Set if an arithmetic overflow occurred. Not affected otherwise.
- E Set if the MAE is used. Cleared otherwise.
- SL Set if the contents of the ACC is automatically saturated. Not affected otherwise.

Addressing Modes	Mnemonic		Rep	Format	Bytes	
-	CoMACM	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm D8 rrrr:qqqq	4	
	CoMACM-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm E8 rrrr:qqqq	4	
	CoMACM	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd	Yes	93 Xm D9 rrrr:qqqq	4	
	CoMACMR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm F8 rrrr:qqqq	4	
	CoMACMR	$[IDX_i\otimes],[Rw_m\otimes],rnd$	Yes	93 Xm F9 rrrr:qqqq	4	
Examples	CoMACM [I	DX1+QX0],[R1+QR1], rnd	; (ACC)∢ ; (R10) ∢ ; ( ((IDX1)	–(ACC)+((IDX0))*((R10)) – (R10) + (QR1) I)-(QX0)) ) ← ((IDX1)) ← (IDX1) + (QX0)	+rnd	
	Repeat 3 times	CoMACM [IDX0 - QX0], [R8	3+QR0]			
	·		; (ACC) -	← (ACC) + ((IDX0))*((R8	]	
			; (R8) ←	(R8) + (QR0)		
	; ( ((IDX0) + (QX0)) ) ← ((IDX0))					
			; (IDX0)	← (IDX0) - (QX0)		
	Repeat MRW ti	mes CoMACM- [IDX1+QX1]	, [R7 - QR(	)]		
			; (ACC) ·	← (ACC) - ((IDX1))*((R7)	)	
			; (R7) ←	(R7) - (QR0)		
			; ( ((IDX <sup>,</sup>	I) - (QX1)) ) ← ((IDX1))		
			; (IDX1)	← (IDX1) + (QX1)		

CoMACM(R)u(-)	Unsign. Multiply-Accumulate Parallel Data Move & Optional Round				
Group	Multiply/Multiply-Accumulate Instructions				
Syntax	CoMACMu [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]				
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) + (tmp) & ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$				
Syntax	CoMACMu [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ], rnd				
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) + (tmp) + 0000 \ 8000_H \\ (MAL) \leftarrow 0 \qquad \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$				
Syntax	CoMACMu- [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]				
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) \text{-} (tmp) & ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$				
Syntax	CoMACMRu [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ]				
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) \text{-} (ACC) \end{array} \qquad \qquad ((IDX_i(\text{-}\otimes))) \leftarrow ((IDX_i)) \end{array}$				
Syntax	CoMACMRu [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗], rnd				
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) - (ACC) + 0000 \ 8000_H \\ (MAL) \leftarrow 0 \qquad \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$				
Data Types	DOUBLE WORD				
Result	40-bit unsigned value				
Description	Multiplies the two unsigned 16-bit source operands pointed to by IDX <sub>i</sub> (i=0,1) and Rw <sub>m</sub> (m=0-15), respectively. The obtained unsigned 32-bit product is first zero-extended, it is then optionally negated prior being added/ subtracted to/from the 40-bit ACC register content, finally the obtained result is optionally rounded before being stored in the 40-bit ACC register. "-" option is used to negate the specified product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and performs two parallel memory reads. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDX <sub>i</sub> overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX <sub>i</sub> , as illustrated by the following table				

Addressing Mode	Overwritten Address
[IDX <sub>i</sub> ]	(no change)
[IDX <sub>i</sub> +]	(IDX <sub>i</sub> )- 2
[IDX <sub>i</sub> -]	(IDX <sub>i</sub> ) + 2
[IDX <sub>i</sub> +QX <sub>j</sub> ]	(IDX <sub>i</sub> ) - (QX <sub>j</sub> )
[IDX <sub>i</sub> -QXj]	$(IDX_i) + (QX_j)$

MAC Condition			_		_			
Flags	N	Z	С	SV	E	SL		
	*	*	*	*	*	*		
	Ν	Set wise	if the mo	st signific	cant bit o	of the res	sult is set. Cleare	d other-
	Z	Set i	if the res	ult equals	zero. C	leared ot	herwise.	
	С	Set i	if a carry	or borrov	v is gene	erated. C	leared otherwise.	
	SV	Set i	if an arith	metic ove	erflow od	curred. I	Not affected other	wise.
	Е	Set i	if the MA	E is used	. Cleare	d otherw	ise.	
	SL	Set affeo	if the co cted othe	ntents of rwise.	the AC	C is auto	omatically satura	ted. Not
<b>Addressing Modes</b>	Mnemon	ic			Rep	Fo	rmat	Bytes
	CoMAC	∕lu [IE	0X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 18 rrrr:rqqq	4
	CoMAC	/lu- [IC	0X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 28 rrrr:rqqq	4
	CoMAC	∕lu [IE	X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗], rno	d Yes	93	Xm 19 rrrr:rqqq	4
	CoMAC	/Ru [IC	0X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 38 rrrr:rqqq	4
	CoMAC	/IRu [ID	א <sub>i</sub> ⊗], [Rי	w <sub>m</sub> ⊗], rno	d Yes	93	Xm 39 rrrr:rqqq	4
Examples	CoMACM	u [IDX′	1+QX0], [F	R10+QR1],	rnd ; (/ ; (I ; [I ; (I	ACC)←(A( R10) ← (R DX1-2.QX DX1) ← (I	CC) + ((IDX0)) * ((R 10) + (QR1) 0] ← [IDX1] DX1) + (QX0)	10))+ rnd
	Repeat 3	times Co	MACMu [I	DX0 - QX0	)], [R8+Q	R0]	, , , ,	
					; (/	ACC) ←(A	CC) + ((IDX0))*((R8	3))
					; (I	R8) ← (R8	3) + (QR0)	
					; (	((IDX0) +	$(QX0))) \leftarrow ((IDX0))$	
					; (I	DX0) ←(I	DX0) - (QX0)	
	Repeat M	RW times	s CoMACI	MRu [IDX1	+QX1], [F	R7 - QR0]		
					; (/	4CC) ←((I >7) ( ) (D7	DX1))^((R7)) - (ACC () (OB0)	)
					, (I • (	$(( \Box X1) \leftarrow (\Box X)$	) - (QNU) (OX1)) ) ∠((IDX1))	
					; (	DX1) ← (I	DX1) + (QX1)	

CoMACM(R)su(-)	Mix. Multiply-Accumulate Parallel Data Move and Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMACMsu [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_{i}))^*((Rw_{m})) \\ (ACC) \leftarrow (ACC) + (tmp) \end{array} \qquad \qquad ((IDX_{i}(\text{-}\otimes))) \leftarrow ((IDX_{i})) \end{array}$
Syntax	CoMACMsu [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ], rnd
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) + (tmp) + 0000\ 8000_H \\ (MAL) \leftarrow 0 \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMsu- [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) \text{-} (tmp) & ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMRsu [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) \text{-} (ACC) \end{array} \qquad \qquad ((IDX_i(\text{-}\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMRsu [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ], rnd
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) - (ACC) + 0000\ 8000_H \\ (MAL) \leftarrow 0 \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$
Data Types	DOUBLE WORD
Result	40-bit signed value
Description	Multiplies the two signed and unsigned 16-bit source operands pointed to by IDX <sub>i</sub> (i=0,1) and Rw <sub>m</sub> (m=0-15), respectively. The obtained signed 32-bit product is first sign-extended, it is then optionally negated prior being added/ subtracted to/from the 40-bit ACC register content, finally the obtained result is optionally rounded before being stored in the 40-bit ACC register. "-" option is used to negate the specified product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and performs two parallel memory reads. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDX <sub>i</sub> overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX <sub>i</sub> , as illustrated by the following table

Addressing Mode	Overwritten Address
[IDX <sub>i</sub> ]	(no change)
[IDX <sub>i</sub> +]	(IDX <sub>i</sub> ) - 2
[IDX <sub>i</sub> -]	(IDX <sub>i</sub> ) + 2
[IDX <sub>i</sub> +QX <sub>j</sub> ]	(IDX <sub>i</sub> ) - (QX <sub>j</sub> )
[IDX <sub>i</sub> - QXj]	$(IDX_i) + (QX_j)$

MAC Condition												
Flags		Ν	Z	С	SV	E	SL	7				
-		*	*	*	*	*	*					
	Ν		Se wis	t if the mose.	ost signifi	cant bit	of the re	sult is set. Cleare	ed other-			
	Ζ		Se	t if the res	sult equals	s zero. C	leared o	therwise.				
	С		Se	t if a carry	or borrov	w is gen	erated. C	leared otherwise				
	S١	/	Se	t if an aritl	hmetic ov	erflow o	ccurred.	Not affected othe	rwise.			
	Е		Se	t if the MA	AE is used	d. Cleare	ed otherw	vise.				
	SL	-	Se aff	t if the co	ontents of erwise.	f the AC	C is aut	omatically satura	ited. Not			
Addressing Modes	Mı	nemo	nic			Rep	p Fo	rmat	Bytes			
	Сс	MAC	Msu [	IDX <sub>i</sub> ⊗], [R	αw <sub>m</sub> ⊗]	Yes	93	Xm 98 rrrr:rqqq	4			
	CoMACMsu- [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]						s 93	Xm A8 rrrr:rqqq	4			
	Сс	MAC	Msu [	IDX <sub>i</sub> ⊗], [R	tw <sub>m</sub> ⊗], rn	d Yes	s 93	Xm 99 rrrr:rqqq	4			
	Сс	MAC	MRsu[	IDX <sub>i</sub> ⊗], [R	[w <sub>m</sub> ⊗]	Yes	s 93	Xm B8 rrrr:rqqq	4			
	Сс	MAC	MRsu[	IDX <sub>i</sub> ⊗], [R	αw <sub>m</sub> ⊗], rn	d Yes	93	Xm B9 rrrr:rqqq	4			
Example	Co	MACN	/Isu [ID]	X1+QX0], [	R10+QR1]	, rnd ;( ;( ;( ;(	ACC) ←(A R10) ← (F ((IDX1) -( IDX1) ← (	ACC) + ((IDX1))*((R R10) + (QR1) QX0)) ) ← ((IDX1)) IDX1) + (QX0)	10)) + rnd			
	Re	peat 3	B times C	CoMACMsu	[IDX0 - Q)	X0], [R8+	QR0], rnd	, , ,				
						; ( ; (	ACC) ← ( R8) ← (R	ACC) + ((IDX0))*((R 3) + (QR0)	:8))			
	; ( ((IDX0) + (QX0)) )← ((IDX0)) ; (IDX0) ← (IDX0) - (QX0)											
	Re	peat N	/IRW tim	es CoMAC	MRsu [IDX	(1+QX1],	[R7 - QR0	), rnd				
						; ( ; (	ACC) ← ( R7) ← (R	(IDX1))*((R7)) - (AC 7) - (QR0)	C) + rnd			
						; (	((IDX1)) -	$(QX1))) \leftarrow ((IDX1))$	)			
						; (	$IDX1) \leftarrow ($	IDX1) + (QX1)				

CoMACM(R)us(-)	Mix. Multiply-Accumulate Parallel Data Move and Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMACMus [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) + (tmp) & ((IDX_i(\text{-}\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMus [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ], rnd
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) + (tmp) + 0000 \ 8000_H \\ (MAL) \leftarrow 0 \qquad \qquad ((IDX_i(\text{-}\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMus- [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (ACC) \text{-} (tmp) & ((IDX_i(\text{-}\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMRus [IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) \text{-} (ACC) \end{array} \qquad \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$
Syntax	CoMACMRus [IDX <sub>i</sub> $\otimes$ ], [Rw <sub>m</sub> $\otimes$ ], rnd
Operation	$\begin{array}{l} (tmp) \leftarrow ((IDX_i))^*((Rw_m)) \\ (ACC) \leftarrow (tmp) - (ACC) + 0000 \ 8000_H \\ (MAL) \leftarrow 0 \qquad \qquad ((IDX_i(-\otimes))) \leftarrow ((IDX_i)) \end{array}$
Data Types	DOUBLE WORD
Result	40-bit signed value
Description	Multiplies the two unsigned and signed 16-bit source operands pointed to by IDX <sub>i</sub> (i=0,1) and Rw <sub>m</sub> (m=0-15), respectively. The obtained signed 32-bit product is first sign-extended, it is then optionally negated prior being added/ subtracted to/from the 40-bit ACC register content, finally the obtained result is optionally rounded before being stored in the 40-bit ACC register. "-" option is used to negate the specified product, "R" option is used to negate the accumulator content, and finally "rnd" option is used to round the result using two's complement rounding. The default sign option is "+" and the default round option is "no round". When "rnd" option is used, MAL register is automatically cleared. Note that "rnd" and "-" are exclusive as well as "-" and "R". This instruction might be repeated and performs two parallel memory reads. In parallel to the arithmetic operation and to the two parallel reads, the data pointed to by IDX <sub>i</sub> overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX <sub>i</sub> , as illustrated by the following table

Addressing Mode	Overwritten Address
[IDX <sub>i</sub> ]	(no change)
[IDX <sub>i</sub> +]	(IDX <sub>i</sub> ) - 2
[IDX <sub>i</sub> -]	(IDX <sub>i</sub> ) + 2
[IDX <sub>i</sub> +QX <sub>j</sub> ]	(IDX <sub>i</sub> ) - (QX <sub>j</sub> )
[IDX <sub>i</sub> - QXj]	$(IDX_i) + (QX_j)$

MAC Condition	N	7	c	SV	F	SI		
Flags	IN	*	*	*	*	3L *		
	Ν	Set i wise	f the mo	st signific	cant bit	of the rea	sult is set. Cleare	d other-
	Z	Set i	f the res	ult equals	s zero. C	leared ot	herwise.	
	С	Set i	f a carry	or borrow	v is gen	erated. C	leared otherwise.	
	SV	Set i	f an arith	metic ov	erflow o	ccurred. I	Not affected other	wise.
	Е	Set i	f the MA	E is used	I. Cleare	d otherw	ise.	
	SL	Set affec	if the co ted othe	ntents of rwise.	the AC	C is aut	omatically satura	ted. Not
Addressing Modes	Mnemonic				Rep	o Fo	rmat	Bytes
	CoMACMu	is [ID	X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 58 rrrr:rqqq	4
	CoMACMu	is- [ID	X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 68 rrrr:rqqq	4
	CoMACMu	is [ID	X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗], rne	d Yes	93	Xm 59 rrrr:rqqq	4
	CoMACMF	Rus[ID	X <sub>i</sub> ⊗], [Rv	w <sub>m</sub> ⊗]	Yes	93	Xm 78 rrrr:rqqq	4
	CoMACMF	Rus[ID	א <sub>i</sub> ⊗], [Rי	w <sub>m</sub> ⊗], rno	d Yes	93	Xm 79 rrrr:rqqq	4
Examples	CoMACMus	[IDX1	+QX0], [F	R10+QR1]	, rnd ; ( ; (	ACC) ← (/ R11) ← (F	ACC) + ((IDX0))*((R (11) + (QR1)	10)) +rnd
					, ( · (	((IDX1) - ( IDX1) ← (	$(QX0)) \leftarrow ((IDX1))$	
	Repeat 3 tim	nes Co	MACMus		, ( (0], [R8+(	OR01		
				[	; (	ACC) ← (/	ACC) + ((IDX0))*((R	8))
					; (	R8) ← (R8	3) + (QR0)	
					; (	((IDX0) +	$(QX0)) \rightarrow ((IDX0))$	)
					; (	$IDX0) \leftarrow ($	IDX0) - (QX0)	
	Repeat MRV	V times		MRus [IDX	1+QX1],	[R7 - QR0	], rnd	
					; (	$ACC) \leftarrow (($	(IDX1))*((R7)) - (AC	C) + rnd
					;(	$((IDV1) \leftarrow (K)$		
					, ( : (	(,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	(QX1)) (QX1)	

#### CoADD(2)

CoADD(2)	Add	
Group	32-bit Arithme	etic Instructions
Syntax	CoADD	op1, op2
Operation	$(tmp) \leftarrow (op2)$ $(ACC) \leftarrow (ACC)$	)\(op1) C) + (tmp)
Syntax	CoADD2	op1, op2
Operation	$(tmp) \leftarrow 2 * (o)$ $(ACC) \leftarrow (ACC)$	op2)\(op1) C) + (tmp)
Data Types	DOUBLE WC	RD
Result	40-bit signed	value
Description	Adds a 40-b result in the a	it operand to the transformed to the transformed set of transformed set of the transformed

Adds a 40-bit operand to the 40-bit Accumulator contents and store the result in the accumulator. The 40-bit operand results from the concatenation of the two source operands op1 (LSW) and op2 (MSW) which is then sign-extended. "**2**" option indicates that the 40-bit operand is also multiplied by two prior being added to ACC. When the MS bit of the MCW register is set and when a 32-bit overflow or underflow occurs, the obtained result becomes 00 7ffff ffff<sub>h</sub> or ff 8000 0000<sub>h</sub>, respectively. This instruction is repeatable with indirect addressing modes and allows up to two parallel memory reads

MAC Condition		_	-		_						
Flags	N	Z	С	SV	E	SL					
	*	*	*	*	*	*					
	Ν	Set if the most significant bit of the result is set. Clear wise.									
	Z	Set	if the res	sult equal	s zero. C	leared o	otherwise.				
	С	Set	if a carry	/ is gener	ated. Cle	eared oth	nerwise.				
	SV	Set	if an arit	hmetic ov	erflow o	ccurred.	Not affected otherwise.				
	Е	Set	if MAE is	s used. C	leared o	therwise					
SL Set if the contents of the ACC affected otherwise.						is auton	natically saturated. Not				
Note	The E-fl The SV-	ag is set flag is s	when th et, when	ne nine hi a 40-bit	ghest bit arithmeti	s of the a c overflo	accumulator are not equal w/ underflow occurs.				

- - -

....



Addressing Modes	Mnemonic		Repeata	RepeatableFormat			
	CoADD	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 02 00	4		
	CoADD2	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 42 00	4		
	CoADD	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 02 rrrr:rqqq	4		
	CoADD2	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 42 rrrr:rqqq	4		
	CoADD	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 02 rrrr:rqqq	4		
	CoADD2	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 42 rrrr:rqqq	4		
Examples	CoADD	R0, R1	; (ACC)	← (ACC) + (R1)\(R0)			
	CoADD2	R2, [R6+]	; (ACC) ← (ACC) + 2*( ((R6))\(R2) ) · (R6) ← (R6) + 2				
	Repeat 3 time	s CoADD [IDX1+QX1], [R10-	+QR0]	()			
			; (ACC) ← (ACC) + ( ((R10))\((II ; (R10) ← (R10) + (QR0)		X1)) )		
	; (IDX1) ← (IDX1) + ( Repeat MRW times CoADD2 R4, [R8 - QR1] <i>;</i> (ACC) ← (ACC) + 2 ; <i>(</i> R8) ← (R8) - (QR1						

## **Addition Examples**

Instr.	MS	op 1	op 2	ACC (before)	ACC (after)	N	Z	С	sv	Е	SL
CoADD	х	0000 <sub>H</sub>	$FFFF_{H}$	00 0100 0000 <sub>H</sub>	00 00FF 0000 <sub>H</sub>	0	0	1	-	0	-
CoADD2	х	0000 <sub>H</sub>	0200 <sub>H</sub>	00 0300 0000 <sub>H</sub>	00 0700 0000 <sub>H</sub>	0	0	0	-	0	-
CoADD	0	0000 <sub>H</sub>	4000 <sub>H</sub>	$7FBFFFFFF_{H}$	$7F FFFF FFFF_{H}$	0	0	0	-	1	-
CoADD	0	0001 <sub>H</sub>	4000 <sub>H</sub>	$7FBFFFFFF_{H}$	80 0000 0000 <sub>H</sub>	1	0	0	1	1	-
CoADD	0	$FFFF_{H}$	$FFFF_{H}$	$FF FFFF FFFF_{H}$	FF FFFF FFFE <sub>H</sub>	1	0	1	-	0	-
CoADD	0	$FFFF_{H}$	$FFFF_{H}$	00 0000 0001 <sub>H</sub>	00 0000 0000 <sub>H</sub>	0	1	1	-	0	-
CoADD	0	$FFFF_{H}$	$FFFF_{H}$	80 0000 0000 <sub>H</sub>	$7F FFFF FFFF_{H}$	0	0	1	1	1	-
CoADD2	0	0001 <sub>H</sub>	2000 <sub>H</sub>	FF C000 0001 <sub>H</sub>	00 0000 0003 <sub>H</sub>	0	0	1	-	0	-
CoADD2	0	0001 <sub>H</sub>	1800 <sub>н</sub>	FF C000 0001 <sub>H</sub>	FF F000 0003 <sub>H</sub>	1	0	0	-	0	-
CoADD	0	B4A1 <sub>H</sub>	73C2 <sub>H</sub>	00 7241 A0C3 <sub>H</sub>	00 E604 5564 <sub>H</sub>	0	0	0	-	1	-
	1				$00.7FFFFFF_{H}$	0	0	0	-	0	1
CoADD	0	B4A1 <sub>H</sub>	A3C2 <sub>H</sub>	FF 8241 A0C3 <sub>H</sub>	FF 2604 5564 <sub>H</sub>	1	0	1	-	1	-
	1				FF 8000 0000 <sub>H</sub>	1	0	1	-	0	1
CoADD	0	B4A1 <sub>H</sub>	73C2 <sub>H</sub>	7F B241 A0C3 <sub>H</sub>	80 2604 5564 <sub>H</sub>	1	0	0	1	1	-
CoADD	0	B4A1 <sub>H</sub>	A3C2 <sub>H</sub>	80 0241 A0C3 <sub>H</sub>	7F A604 5564 <sub>H</sub>	0	0	1	1	1	-

# CoSUB(2)(R)

CoSUB(2)(R)	Subtract							
Group	32-bit Arithmetic Instructions							
Syntax	CoSUB	ор	1, op2					
Operation	$(tmp) \leftarrow (op)$ $(ACC) \leftarrow (A)$	02)\(op1 .CC) - (	1) (tmp)					
Syntax	CoSUB2	ор	1, op2					
Operation	$(tmp) \leftarrow 2 *$ $(ACC) \leftarrow (A)$	(op2)\( .CC) - (	(op1) (tmp)					
Syntax	CoSUBR	ор	1, op2					
Operation	$(tmp) \leftarrow (op)$ $(ACC) \leftarrow (tr)$	02)\(op1 mp) - (A	1) \CC)					
Syntax	CoSUB2R	ор	1, op2					
Operation	$(tmp) \leftarrow 2 *$ $(ACC) \leftarrow (tr$	(op2)\( np) - (A	(op1) ACC)					
Data Types	DOUBLE W	/ORD						
Result	40-bit signe	d value	9					
Description	Subtracts a versa when bit operand (LSW) and the 40-bit of from/to ACC when a 32-b 7ffff ffff <sub>h</sub> or indirect add	40-bit "r" opti- results op2 (M- operanc C/negat oit over ff 8000 ressing	opera on is us from t SW) wi d is als ted ACC flow or 0 0000 g mode	nd from sed and s the conca hich is th so multip C. When underflo h, respects s and allo	the 40- store the atenation en sign- lied by the the MS w occurs ctively. T ows up t	bit Accur result in n of the t extended two prior bit of the s, the obt his instru- o two par	mulator contents or vice the accumulator. The 40- wo source operands op1 I. " <b>2</b> " option indicates that being subtracted/added MCW register is set and ained result becomes 00 uction is repeatable with rallel memory reads	
MAC Condition								
Flags	N *	<b>Z</b>	<b>C</b>	SV *	<b>E</b>	SL *		
	N	Set if t wise.	the mo	st signifio	cant bit o	of the res	sult is set. Cleared other-	
	Z	Set if t	he resu	ult equals	s zero. C	leared ot	herwise.	
	С	Set if a	a borrov	w is gene	erated. C	leared of	herwise.	
	SV	Set if a	an arith	metic ov	erflow oo	ccurred.	Not affected otherwise.	
	E	Set if t		E IS USEC	the AC	a otherw	ISE.	
	JL	affecte	d other	wise.			mancany salurateu. Nol	
Note	The E-flag is The SV-flag	s set w is set,	hen the when a	e nine hig a 40-bit a	hest bits arithmetio	s of the a c overflov	ccumulator are not equal. v/ underflow occurs.	

Addressing Modes	Mnemonic		Rep	Format	Bytes
-	CoSUB	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 0A 00	4
	CoSUBR	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 12 00	4
	CoSUB2	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 4A 00	4
	CoSUB2R	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 52 00	4
	CoSUB	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 0A rrrr:rqqq	4
	CoSUBR	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 12 rrrr:rqqq	4
	CoSUB2	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 4A rrrr:rqqq	4
	CoSUB2R	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	Yes	93 Xm 52 rrrr:rqqq	4
	CoSUB	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 0A rrrr:rqqq	4
	CoSUBR	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 12 rrrr:rqqq	4
	CoSUB2	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 4A rrrr:rqqq	4
	CoSUB2R	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	Yes	83 nm 52 rrrr:rqqq	4
Examples	CoSUB I	R0, R1	; (ACC)	← (ACC) - (R1)\(R0)	
	CoSUB2	R2, [R6+]	; (ACC)	← (ACC) + 2*( ((R6))\ (R	2))
			; (R6) ←	(R6) + 2	
	Repeat 3 times	CoSUB [IDX1+QX1], [R10+	QR0]		
			; (ACC)	← (ACC) - ( ((R10))\((ID>	(1))))
			; (R10) (	-(R10) + (QR0)	
	Roppot MRW/t		; (IDX1) 11 : (ACC)	$\leftarrow (IDX1) + (QX1)$	
	Repeat WRW t	1111250030012 K4, [K8 - QK	· (R8) ∠	← ∠ (((Kō))\(K4) )- (AUU . (R8) - (OR1)	<i>•</i> )
			, (10)		

# **Subtraction Examples**

Instr.	MS	op 1	op 2	ACC (before)	ACC (after)	N	z	С	sv	Е	SL
CoSUB	x	183A <sub>H</sub>	72AC <sub>H</sub>	$00.7FFFFFF_{H}$	00 0D53 E7C5 <sub>H</sub>	0	0	0	-	0	-
CoSUBr	x	183А <sub>н</sub>	$72AC_{H}$	00 7FFF FFFF <sub>H</sub>	FF F2AC 183B <sub>H</sub>	1	0	1	-	0	-
CoSUB2	x	0C1D <sub>H</sub>	3956 <sub>н</sub>	00 E604 5564 <sub>H</sub>	00 7358 3D2A <sub>H</sub>	0	0	0	-	0	-
CoSUBr2	x	0C1D <sub>H</sub>	3956 <sub>н</sub>	00 E604 5564 <sub>H</sub>	FF 8CA7 C2D6 <sub>H</sub>	1	0	1	-	0	-
CoSUB	0	FFFF <sub>H</sub>	FFFF <sub>H</sub>	$7FFFFFFFF_{H}$	80 0000 0000 <sub>H</sub>	1	0	1	1	1	-
	1				$007FFFFFF_{H}$	0	0	1	1	0	1
CoSUB2	0	0000 <sub>H</sub>	3000 <sub>H</sub>	$7F FFFF FFFF_{H}$	$7F 9FFF FFFF_{H}$	0	0	0	-	1	-
CoSUB2	0	0001 <sub>H</sub>	0000 <sub>H</sub>	80 0000 0000 <sub>H</sub>	7F FFFF FFFE <sub>H</sub>	0	0	0	1	1	-
	1				FF 8000 0000 <sub>H</sub>	1	0	0	1	0	1

### CoNEG

CoNEG	Negate A	ccum	ulator	with Op	otional	Roundi	ing	
Group	32-bit Arith	metic lı	nstructio	ons				
Syntax	CoNEG CoNEG	rn	d					
Operation	$\begin{array}{l} IF \ (rnd) \ TH \\ (ACC) \leftarrow 0 \\ (MAL) \leftarrow 0 \\ ELSE \\ (ACC) \leftarrow 0 \\ END \ IF \end{array}$	EN - (ACC - (ACC	\$) + 000 \$)	08000 <sub>H</sub>				
Data Types	ACCUMUL	ATOR						
Result	40-bit signe	ed value	е					
Description	The Accum rounded be MAL is clea bit overflow ff 8000 000	efore be ared. W or unc 0 <sub>h</sub> , resj	content eing sto /hen the derflow o pectivel	is subtra pred in the MS bit poccurs, t y. This in	acted from he accur of the MC he obtain hstruction	m zero a nulator i CW regis ned resu is not re	and the result register. with ster is set and It becomes 00 epeatable	is optionally 'rnd" option when a 32- ) 7ffff ffff <sub>h</sub> or
MAC Condition Flags	N *	<b>Z</b>	<b>C</b> *	SV *	<b>E</b> *	SL *		
	N	Set if wise.	the mos	st signifi	cant bit c	of the re	sult is set. Cle	eared other-
	Z	Set if	the resu	It equals	s zero. Cl	eared of	herwise.	
	С	Set if a	a borrov	v is gene	erated. C	leared o	therwise.	
	SV	Set if a	an arith	metic ov	erflow oc	curred.	Not affected o	therwise.
	E	Set if	the MAE	E is used	d. Cleare	d otherw	ise.	
	SL	Set if affecte	the cor ed other	ntents of wise.	the AC	C is aut	omatically sat	urated. Not
Addressing Modes	Mnemonic				Rep	Fo	rmat	Bytes
	CoNEG				No	A3	00 32 00	4
	CoNEG	rnd			No	A3	00 72 00	4
Examples	CoNEG CoNEG	rnd			; (A ; (A	0 → (CC) \CC) ← 0	- (ACC) - (ACC) + rnd	

Instr.	MS	rnd	ACC (before)	ACC (after)	N	Ζ	С	SV	Е	SL
CoNEG	х	No	00 1234 5678 <sub>н</sub>	FF EDCB A988 <sub>H</sub>	1	0	0	-	0	-
CoNEG	х	Yes	00 1234 5678 <sub>н</sub>	FF EDCC 0000 <sub>H</sub>	1	0	0	-	0	-

CoABS	Absolute	Valu	е					
Group	32-bit Arith	metic	Instructio	ons				
Syntax	CoABS							
Operation	$(ACC) \leftarrow A$	bs( A	CC)					
Syntax	CoABS	c	op1, op2					
Operation	$(ACC) \leftarrow A$	vps( (o	p2)\(op1	))				
Data Types	ACCUMUL	ATOR	, DOUBI	E WOR	D			
Result	40-bit sign	ed valu	ue					
Description	Compute the or the absorbed Accumulate source operative operations instructed to the source operation of the source operation	ne abs plute v pr. The erands ption is	solute val value of a e 40-bit o s op1 (LS s not rep	lue of the a 40-bit operand SW) and eatable	e Accum source o results f op2 (M	ulator operan from th SW) w	if no operands are s d and load the resu ne concatenation of hich is then sign-e	pecified It in the the two stended.
MAC Condition								
	N	7	C	sv	F	SI		
Flags	N *	<b>Z</b> *	<b>C</b>	SV -	E *	SL *		
Flags	N *	× Set i wise.	C 0 f the mo	SV - st signifie	E * cant bit	SL * of the	result is set. Cleare	d other-
Flags	N * N Z	× Set if wise. Set if	C 0 f the mo	SV - st signific	E * cant bit of s zero. C	SL * of the	result is set. Cleare	d other-
Flags	N X N Z C	z * Set i Set i Alwa	C 0 f the mode f the result ys cleare	SV - st signific ult equals ed.	E * cant bit o s zero. C	SL * of the	result is set. Cleare otherwise.	d other-
Flags	N X N Z C SV	z * Set i Set i Alwa Not a	C 0 f the move f the result ys cleare affected.	SV - st signifie ult equals ed.	E * cant bit o s zero. C	SL * of the	result is set. Cleare otherwise.	d other-
Flags	N Z C SV E	x Set if Set if Alwa Not a Set if	C 0 f the model f the result ys cleare affected. f the MAI	SV - st signific ult equals ed. E is usec	E * cant bit o s zero. C d. Cleare	SL * of the cleared	result is set. Cleare l otherwise. erwise.	d other-
Flags	N Z C SV E SL	x Set if Set if Alwa Not a Set if Set if affect	C 0 f the model f the result ys cleare affected. f the MAI f the contect f the contect	SV - st signific alt equals ed. E is usec ntents of rwise.	E * cant bit s zero. C d. Cleare	SL * of the cleared ed othe cC is a	result is set. Cleare otherwise. erwise. automatically satura	d other- ted. Not
Addressing Modes	N Z C SV E SL Mnemonic CoABS	x Set if Set if Alwa Not a Set if Set if affect	C 0 f the model f the result ys cleared affected. f the MAI f the contect f the contect f the contect f the contect f the result f the contect f the result f the contect f f the contect f f f f f f f f f f f f f f f f f f f	SV - st signifie ult equals ed. E is used ntents of rwise.	E * cant bit of s zero. C I. Cleare the AC Rep No	SL * of the cleared cd othe cC is a	result is set. Cleare otherwise. automatically satura Format A3 00 1A 00	d other- ted. Not Bytes 4
Addressing Modes	N Z C SV E SL Mnemonic CoABS CoABS	x Set if Set if Alwa Not a Set if Set i affec	C 0 f the model f the result ys cleared affected. f the MAI f the Coll f the coll f the	SV - st signific ult equals ed. E is used ntents of wise.	E * cant bit of s zero. C d. Cleare the AC Rep No No	SL * of the cleared cd othe cC is a	result is set. Cleare otherwise. automatically satura Format A3 00 1A 00 A3 nm CA 00	d other- ted. Not Bytes 4 4
Addressing Modes	N * N Z C SV E SL Mnemonic CoABS CoABS CoABS	x Set if Set if Alwa Not a Set if Set if affec	C 0 f the model f the result of the result ys cleared affected. f the MAI f the Contect f the MAI f the contect $V_n$ , $\mathbb{R}w_m$ $X_i \otimes ]$ , $[\mathbb{R}w_m]$	SV - st signifie ult equals ed. Ξ is usec ntents of rwise. V <sub>m</sub> ⊗]	E * cant bit of s zero. C d. Cleare the AC Rep No No	SL * of the cleared cd othe cC is a	result is set. Cleare otherwise. automatically satura Format A3 00 1A 00 A3 nm CA 00 93 Xm CA 0:0qqq	d other- ted. Not Bytes 4 4 4

## CoLOAD(2)(-)

CoLOAD(2)(-)	Load Aco	cumu	lator				
Group	32-bit Arith	metic	Instructio	ons			
Syntax	CoLOAD	c	p1, op2				
Operation	$(tmp) \leftarrow (o)$ $(ACC) \leftarrow 0$	p2)\(o ) + (tm	p1) p)				
Syntax	CoLOAD-	С	p1, op2				
Operation	$(tmp) \leftarrow (o)$ $(ACC) \leftarrow 0$	p2)\(o ) - (tmp	p1) o)				
Syntax	CoLOAD2	С	p1, op2				
Operation	$(tmp) \leftarrow 2^*$ $(ACC) \leftarrow 0$	( (op2) ) + (tm	)\(op1)) p)				
Syntax	CoLOAD2-	- c	p1, op2				
Operation	$(tmp) \leftarrow 2^*$ $(ACC) \leftarrow 0$	( (op2) ) - (tmp	)\(op1)) >)				
Data Types		NORD					
Result	40-bit sign	ed valu	he				
Description	Loads the operand re (LSW) and indicate th respectivel that the so register is result becc not repeat	accur esults d op2 at the y, prio urce o set an omes ( able ar	mulator w from the (MSW) 40-bit o r being s perand is d when a 00 7ffff fff nd allows	with a 4 e concate which is perand i stored in s 2's con a 32-bit o $f_h$ or ff 8 up to tw	0-bit sou enation of then si is also n the acco nplement overflow 000 0000 vo paralle	urce ope of the tw gn-exten nultiplied umulator ted. Whe or under O <sub>h</sub> , respe el memor	erand. The 40-bit source vo source operands op1 ided. "2" and "-" options I by two or/and negated, . The "-" option indicates en the MS bit of the MCW flow occurs, the obtained ectively. This instruction is ry reads.
MAC Condition		_	•	<b>.</b>	_		
Flags	N *	Ζ *	С *	SV	<b>E</b> *	SL *	
	N	Set it wise.	f the mos	st signific	cant bit o	of the res	sult is set. Cleared other-
	Z	Set if	f the resu	It equals	s zero. C	leared ot	therwise.
	С	Set if	f a borrov	v is gene	erated. C	leared o	therwise.
	SV	Not a	affected.				
	E	Set if	the MAE	E is usec	I. Cleare	d otherw	ise.
	SL	Set i affec	t the cor ted other	ntents of wise.	the AC	C is aut	omatically saturated. Not



## CoLOAD(2)(-)

Addressing Modes	Mnemonic		Rep	Format	Bytes
	CoLOAD	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 22 00	4
	CoLOAD-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 2A 00	4
	CoLOAD2	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 62 00	4
	CoLOAD2-	Rw <sub>n</sub> , Rw <sub>m</sub>	No	A3 nm 6A 00	4
	CoLOAD	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 22 0:0qqq	4
	CoLOAD-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 2A 0:0qqq	4
	CoLOAD2	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 62 0:0qqq	4
	CoLOAD2-	[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]	No	93 Xm 6A 0:0qqq	4
	CoLOAD	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 22 0:0qqq	4
	CoLOAD-	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 2A 0:0qqq	4
	CoLOAD2	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 62 0:0qqq	4
	CoLOAD2-	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]	No	83 nm 6A 0:0qqq	4

CoNOP	No-Ope	ration									
Group	32-bit Arit	hmetic	Instructio	ons							
Syntax	CoNOP	CoNOP									
Operation	No Opera	tion									
Description	Modifies t ters.	he add	ress poir	nters with	out char	iging the	e internal MAC-Ur	it regis-			
MAC Condition	N	z	С	sv	Е	SL					
T lags	-	-	-	-	-	-	]				
	N	Not	affected.								
	Z	Not	affected.								
	С	Not	affected.								
	SV	Not	affected.								
	E	Not	affected.								
	SL	Not	affected.								
Addressing Modes	Mnemoni	C			Rep	Fo	rmat	Bytes			
	CoNOP	[R\	v <sub>m</sub> ⊗]		Yes	93	0m 5A rrrr:rqqq	4			
	CoNOP	[ID	X <sub>i</sub> ⊗]		Yes	93	X0 5A rrrr:r000	4			
	CoNOP	[ID	X <sub>i</sub> ⊗], [Rv	v <sub>m</sub> ⊗]	Yes	93	Xm 5A rrrr:rqqq	4			
Examples	CoNOP	[IDX0	+QX1], [R	11+QR1]	; (F ; (II	R11) ← (F DX0) ← (	R11) + (QR1) IDX0) + (QX1)				
	CoNOP	[R1+]			; (F	R1) ← (R1	1) + 2				
	Repeat MR	W times	CoNOP [	IDX1+QX0	)] ; (II	OX1) ← (	IDX1) + (QR0)				



Syntax	CoSHL	op1						
Operation	$(count) \leftarrow (C) \leftarrow 0$ DO WHI $(C) \leftarrow (A)$ $(ACCn) \leftarrow (ACCn) \leftarrow (ACC0) \leftarrow (Count) \leftarrow (Count) \leftarrow (ACC0)$	– (op1) LE (count) ≠ \CC39) ← (ACCn-1) ← 0 – (count) -1 HLE	0 [n=1:	39]				
Data Types	ACCUM	ULATOR						
Result	40-bit sig	gned value						
Description	Shifts th op1. The values c unsigned unsigned When th underflow resp. Thi	e ACC regis e least signif ontained bet d immediate d data) of a e MS bit of t w occurs, the is instruction	ter left by icant bits ween 0 ar data, or ny registe he MCW obtained is repeata	as m of the nd 8 a the 1 er dire regist resul ble w	hany tin e resu are all least s ectly c ter is s lt beco hen "o	mes as lt are fil owed. " significa or indire set and mes 00 p 1" is r	specified by the led with zeros. O op1" can be eithe nt 4 bits (consid ectly addressed o when a 32-bit ove 7ffff ffff <sub>h</sub> or ff 800 not an immediate o	operand nly shift r a 4-bit ered as operand. erflow or 0 0000 <sub>h</sub> , operand.
MAC Condition								
Flags	N *	Z (	S SV	, 	E *	SL *	]	
	N	Set if the mo	st signific	ant bi	t of the	e result	is set. Cleared oth	erwise.
	N Z	Set if the mo Set if the res	st signific	ant bi zero.	t of the . Clear	e result ed othe	is set. Cleared oth rwise.	erwise.
	N Z C	Set if the mo Set if the res The carry fla	est significa sult equals ag is set ac	ant bi zero. ccordi	t of the . Clear ing to t	e result ed othe he last	is set. Cleared oth rwise. MSB shifted out o	erwise. f op1.
	N Z C SV	Set if the mo Set if the res The carry fla Set if the las	est significa sult equals og is set ac t shifted o	ant bi zero. ccordi ut bit	t of the . Clear ing to t is diffe	e result ed othe he last erent fro	is set. Cleared oth rwise. MSB shifted out o m N.	erwise. f op1.
	N Z C SV E	Set if the mo Set if the res The carry fla Set if the las Set if the MA	est significa sult equals og is set ac t shifted o NE is used	ant bi zero ccordi ut bit . Clea	t of the . Clear ing to t is diffe ared ot	e result ed othe he last erent fro herwise	is set. Cleared oth rwise. MSB shifted out o m N. s.	erwise. f op1.
	N Z C SV E SL	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the c affected of	est signification of the equals of the states of the shifted of the shifted of the states of the sta	ant bi zero. ccordi ut bit . Clea the	t of the . Clear ing to t is diffe ared ot ACC	e result red othe the last erent fro herwise is aut	is set. Cleared oth rwise. MSB shifted out o m N. e. omatically saturat	erwise. f op1. ed. Not
Addressing Modes	N Z C SV E SL Mnemon	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the MA Set if the c affected o	est significa gult equals g is set ac t shifted o t shifted o t is used ontent of otherwise.	ant bi ccordi ut bit . Clea the	t of the . Clear ing to t is diffe ared ot ACC Rep	e result ed othe he last erent fro herwise is aut Fc	is set. Cleared oth rwise. MSB shifted out o m N. s. omatically saturat	erwise. f op1. ed. Not Bytes
Addressing Modes	N Z C SV E SL Mnemon CoSHL	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the MA Set if the c affected o nic Rw <sub>n</sub>	est significa sult equals of is set ac t shifted o t is used ontent of otherwise.	ant bi ccordi ut bit . Clea the	t of the . Clear ing to t is diffe ared ot ACC Rep Yes	e result red othe the last erent fro therwise is aut Fc A3	is set. Cleared oth rwise. MSB shifted out o m N. e. omatically saturat ormat 3 nn 8A rrrr:r000	erwise. f op1. red. Not Bytes 4
Addressing Modes	N Z C SV E SL Mnemon CoSHL CoSHL	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the MA Set if the c affected o nic Rwn #data4	est significa g is set ac t shifted o t is used ontent of otherwise.	ant bi ccordi ut bit . Clea the	t of the . Clear ing to t is diffe ared ot ACC Rep Yes No	e result ed othe he last erent fro herwise is aut Fc A3	is set. Cleared oth rwise. MSB shifted out o m N. s. omatically saturat ormat 3 nn 8A rrrr:r000 3 00 82 ssss:0	erwise. f op1. eed. Not Bytes 4 4
Addressing Modes	N Z C SV E SL Mnemon CoSHL CoSHL CoSHL	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the MA Set if the c affected o nic Rw <sub>n</sub> #data <sub>4</sub> [Rw <sub>m</sub> ⊗	est significa g is set ac t shifted o t is used ontent of otherwise.	ant bi zero. ccordi ut bit . Clea the	t of the . Clear ing to t is diffe ared ot ACC Rep Yes No Yes	e result ed othe he last erent fro herwise is aut Fo A3 83	is set. Cleared oth rwise. MSB shifted out o m N. e. omatically saturat ormat 3 nn 8A rrrr:r000 3 00 82 ssss:0 5 mm 8A rrrr:rqqq	erwise. f op1. red. Not Bytes 4 4 4
Addressing Modes	N Z C SV E SL Mnemon CoSHL CoSHL CoSHL CoSHL CoSHL	Set if the mo Set if the res The carry fla Set if the las Set if the MA Set if the MA Set if the c affected o nic Rwn #data <sub>4</sub> [Rwm⊗ #3 R3	est significa gult equals g is set ac t shifted o E is used ontent of otherwise.	ant bi zero. ccordi ut bit . Clea the	t of the Clear ing to t is diffe ared ot ACC Rep Yes No Yes ; (A ; (A	e result ed othe the last erent fro therwise is aut Fc ACC ACC ACC ACC ACC ACC ACC ACC ACC AC	is set. Cleared oth rwise. MSB shifted out o m N. omatically satural ormat 3 nn 8A rrrr:r000 3 00 82 ssss:0 5 mm 8A rrrr:rqqq ACC) << 3 ACC) << (R33-0)	erwise. f op1. red. Not Bytes 4 4 4

Accumulator Logical Shift Left

Shift Instructions

CoSHL

Group

### CoSHR

CoSHR	Accumul	ator I	ogical	Shift I	Right			
Group	Shift Instru	ctions						
Syntax	CoSHR	c	pp1					
Operation	$(count) \leftarrow (c) \leftarrow 0$ DO WHILE $((ACCn) \leftarrow (ACC39) \leftarrow (count) \leftarrow (count) \leftarrow (count)$	op1) (cour (ACC - 0 (count E	nt) ≠ 0 :n+1) [r ) -1	038=1	3]			
Data Types	ACCUMUL	ATOR						
Result	40-bit signe	ed valu	ue					
Description	Shifts the A op1. The n Only shift v a 4-bit unsi unsigned d MS bit of t repeatable	ACC ro nost s alues gned i ata) o the M when	egister ri ignificant containe immediat if any reg CW regi "op 1" is	ght by a bits of d betwe e data, gister di ster do not an	as many f the result een 0 and or the lea rectly or i es not af immediat	times as It are fill 8 are a ast signi ndirectly fect the e opera	s specified by the o ed with zeros acco llowed. "op1" can b ficant 4 bits (consid y addressed opera result. This instru nd.	operand ordingly. le either lered as nd. The liction is
MAC Condition		_	_		_			
Flags	N *	Z *	<b>C</b>	SV -	E *	SL -		
	N	Set i wise	f the mo	st signi	icant bit o	of the re	⊔ esult is set. Cleare	d other-
	Z	Set if	f the resu	ılt equa	ls zero. C	leared o	otherwise.	
	С	Clea	red alwa	/S.				
	SV	Not	affected.					
	E	Set if	f the MAI	E is use	d. Cleare	d other	wise.	
	SL	Not	affected.		_			
Addressing Modes		D.,			Rep	beatable	Format	Bytes
		тм #d	'n ata		No	A	3 III 9A III.1000	4
	CoSHR	#u	ata₄ w <sub>m</sub> ⊗]		Yes	8	3 mm 9A rrrr:rqqq	4
Examples	CoSHR CoSHR CoSHR	#3 R3 [R10	- QR0]		; (, ; (, ; (, ; ()	ACC) ← ACC) ← ACC) ← R10) ← (	(ACC) >> 3 (ACC) >> (R33-0) (ACC) >> ((R103-0)) R10) - (QR0)	

# CoASHR Accumulator Arithmetic Shift Right with Optional Round

Group	Shift Instru	ctions	5				
Syntax	CoASHR CoASHR	0	op1 op1, rnd				
Operation	$\begin{array}{l} (\text{count}) \leftarrow (\\ (C) \leftarrow 0\\ \text{DO WHILE}\\ (\text{ACCn}) \leftarrow (\\ (\text{count}) \leftarrow (\\ \text{END WHIL}\\ \text{IF (rnd) TH}\\ (\text{ACC}) \leftarrow (\\ (\text{MAL}) \leftarrow 0\\ \text{END IF} \end{array}$	(cou (ACC count E EN ACC)	nt) ≠ 0 n+1) [r t) -1 + 000080	n=038] 000 <sub>н</sub>			
Data Types	ACCUMUL	ATOR	R				
Result	40-bit signe	ed val	ue				
Description	Arithmetica the operan cant bits of ones if the allowed. "o significant indirectly a register do set and wh becomes 0 able when	Illy sh d op1 the re origi p1" c 4 bits ddres es not nen a 0 7ffff "op 1"	ifts the Ad . To pres- esult are to inal MSB an be eit a conside sed oper t affect the a 32-bit of f ffff <sub>h</sub> or ff i is not an	CC regis erve the filled with was 1. her a 4- ered as and. Wit e result. werflow 8000 000	ter right sign of Only s bit unsig hout "rn While w or unde D0 <sub>h</sub> , resp ate oper	by as ma the ACC if the orig hift value gned imm d data) o d" option d" option ith "rnd" o erflow occ pectively. and.	any times as specified by register, the most signifi- inal MSB was a 0 or with s between 0 and 8 are ediate data, or the least f any register directly or , the MS bit of the MCW option and if the MS bit is curs, the obtained result This instruction is repeat-
MAC Condition	Ν	7	C	SV	F	<b>CI</b>	
Flags	*	*	*	*	*	*	
	Ν	Set i wise	if the mos	st signific	cant bit	of the res	ult is set. Cleared other-
	Z	Set i	f the resu	lt equals	s zero. C	leared ot	nerwise.
	C	Set i	f a carry i	is genera	ated (rnc	d). Cleare	d otherwise.
	50	wise	if an aritr	imetic o	vertiow	occurred	(rnd). Not affected other-
	E	Set i	f the MAE	E is used	l. Cleare	d otherwi	se.
	SL	Set Not a	if the cor affected c	ntents of otherwise	the AC	C is autor	matically saturated (rnd).

### CoASHR

Addressing Modes	Mnemonic		Rep	Format	Bytes
-	CoASHR	Rw <sub>n</sub>	Yes	A3 nn AA rrrr:r000	4
	CoASHR	Rw <sub>n</sub> , rnd	Yes	A3 nn BA rrrr:r000	4
	CoASHR	#data <sub>4</sub>	No	A3 00 A2 ssss:0	4
	CoASHR	#data <sub>4</sub> , rnd	No	A3 00 B2 ssss:0	4
	CoASHR	[Rw <sub>m</sub> ⊗]	Yes	83 mm AA rrrr:rqqq	4
	CoASHR	[Rw <sub>m</sub> ⊗], rnd	Yes	83 mm BA rrrr:rqqq	4
Examples	CoASHR	#3, rnd	; (ACC)	$\leftarrow$ (ACC) >>a 3 + rnd	
-	CoASHR	R3	; (ACC)	← (ACC) >>a (R33-0)	
	CoASHR	[R10 - QR0]	; (ACC)	$\leftarrow$ (ACC) >>a ((R103-0))	
			; (R10)	← (R10) - (QR0)	



CoRND	Round Accumulator									
Group	Shift Instructions									
Syntax	CoRND									
Operation	$(ACC) \leftarrow (ACC) + 0000 \ 8000_H$ $(MAL) \leftarrow 0$									
Data Types	ACCUMULATOR									
Result	40-bit signed value									
Description	Rounds the ACC register contents by adding 0000 8000h to it and store the result in the ACC register and the lower part of the ACC register, MAL, is cleared. When the MS bit of the MCW register is set and when a 32-bit overflow or underflow occurs, the obtained result becomes 00 7ffff ffff <sub>h</sub> or ff 800 $0000_{h}$ , respectively. This instruction is not repeatable.									
MAC Condition										
Flags	N *	<b>Z</b> *	<b>C</b>	SV *	E *	SL *				
	N	Set i wise.	Set if the most significant bit of the result is set. Cleared otherwise.							
	Z	Set if	Set if the result equals zero. Cleared otherwise.							
	С	Set if	Set if a carry is generated. Cleared otherwise.							
	SV	Set if an arithmetic overflow occurred. Not affected otherwise.								
	E	Set if the MAE is used. Cleared otherwise.								
	SL	Set if the contents of the ACC is automatically saturated. Not affected otherwise.						ated. Not		
Addressing Modes	Mnemoni CoRND	C			Rep No	Fo A3	rmat 00 B2 00	Bytes 4		
Notes	CoRND is	s equiva	alent to C	oASHR	#0, rnd.					
Example	$CoRND \qquad \qquad ; (ACC) \gets (AC$					ACC) + rnd				

#### CoMAX

СоМАХ	Maximum										
Group	Compare Instructions										
Syntax	CoMAX		op1 op2								
Operation	$(tmp) \leftarrow (op2) (op1)$ (ACC) $\leftarrow max((ACC), (tmp))$										
Data Types	DOUBLE WORD										
Result	40-bit signed value										
Description	Compares a signed 40-bit operand against the ACC register content. The 40-bit operand results from the concatenation of the two source operands op1 (LSW) and op2 (MSW) which is then sign-extended. If the contents of the ACC register is smaller than the 40-bit operand, then the ACC register is loaded with it. Otherwise the ACC register remains unchanged. The MS bit of the MCW register does not affect the result. This instruction is repeatable with indirect addressing modes.										
MAC Condition			_								
Flags	N	Z	C	SV	E		SL				
	N Z	Set if the most significant bit of the result is set. Cleared other- wise. Set if the result equals zero. Cleared otherwise.									
	C	C	Cleared always.								
	SV F	N C	Not affected.								
	E SL	Set if the contents of the ACC register is changed. Not affected otherwise.									
Addressing Modes	Mnemon	ic			F	Rep	For	mat	Bytes		
-	CoMAX		Rw <sub>n</sub> , Rw <sub>m</sub>			NO.	A3	nm 3A 00	4		
	CoMAX		[IDX <sub>i</sub> ⊗], [Rw <sub>m</sub> ⊗]			/es	93	Xm 3A rrrr:rqqq	4		
	CoMAX		Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]			/es	83	nm 3A rrrr:rqqq	4		
Examples	CoMAX	( [IDX1+QX0], [R11+QR1]				; (ACC)← Max((ACC),((R11))\((IDX1))) ; (R11) ← (R11) + (QR1) ; (IDX1) ← (IDX1) + (QX0)					
	CoMAX	R1, R1				; (ACC) ← Max( (ACC), (R10)\(R1) )					
	Repeat 23 times CoMAX R5, [R6 - QR0]; (ACC) $\leftarrow$ Max( (ACC), ((R6))\(R5)) ); (R6) $\leftarrow$ (R6) - (QR0)							₹5))))			
CoMIN	Minimum										
------------------	---	--	-----------------------------------	-------------------	-----------	--	------------------------------------	-------	--	--	--
Group	Compare Instructions										
Syntax	CoMIN op1, op2										
Operation	$(tmp) \leftarrow (op2) (op1)$ (ACC) $\leftarrow min((ACC), (tmp))$										
Data Types	DOUBLE WORD										
Result	40-bit signed value										
Description	Compares a signed 40-bit operand against the ACC register content. The 40-bit operand results from the concatenation of the two source operands op1 (LSW) and op2 (MSW) which is then sign-extended. If the contents of the ACC register is greater than the 40-bit operand, then the ACC register is loaded with it. Otherwise the ACC register remains unchanged. The MS bit of the MCW register does not affect the result. This instruction is repeatable with indirect addressing modes.										
MAC Condition		_	-		_						
Flags	N	Z	C	SV	E	SL					
	<ul> <li>N Set if the most significant bit of the result is set. Cleared oth wise.</li> <li>Z Set if the result equals zero. Cleared otherwise.</li> <li>C Cleared always.</li> <li>SV Not affected</li> </ul>										
	Е	Set if the MAE is used. Cleared otherwise.									
	SL	SL Set if the contents of the ACC register is changed. Not affected otherwise.									
Addressing Modes	Mnemonic				Re	ep F	Format	Bytes			
	CoMIN	Rw	/ <sub>n</sub> , Rw <sub>m</sub>		No	b A	A3 nm 7A 00	4			
	CoMIN	[ID	X <sub>i</sub> ⊗], [Rv	v <sub>m</sub> ⊗]	Ye	s g	93 Xm 7A rrrr:rqqq	4			
	CoMIN	Rw	/ <sub>n</sub> , [Rw <sub>m</sub>	8]	Ye	s 8	33 nm 7A rrrr:rqqq	4			
Examples	CoMIN	[IDX1+QX0], [R11+QR1]			;	; (ACC) ← min( (ACC), ((R11))\((IDX1)) ) ; (R11) ← (R11) + (QR1) ; (IDX1) ← (IDX1) + (QX0)					
	CoMIN	CoMIN R1, R10					; (ACC) ← min( (ACC), (R10)\(R1) )				
	Repeat 23 times CoMIN R5, [R6 - QR0]				)] ; ;	; (ACC) ← min( (ACC), ((R6))\(R5)) ) ; (R6) ← (R6) - (QR0)					

**57** 

СоСМР	Compare									
Group	Compare Instructions									
Syntax	CoCMP	CoCMP op1, op2								
Operation	$tmp \leftarrow (op2) (op1)$ $(ACC) \Leftrightarrow (tmp)$									
Data Types	DOUBLE WORD									
Description	Subtracts a 40-bit signed operand from the 40-bit Accumulator content and update the N, Z and C flags contained in the MSW register leaving the accumulator unchanged. The 40-bit operand results from the concatenation, "\", of the two source operands op1 (LSW) and op2 (MSW) which is then signextended. The MS bit of the MCW register does not affect the result. This instruction is not repeatable and allows up to two parallel memory reads.									
MAC Condition		_								
Flags	N *	Z *	<b>C</b> *	SV -	E	=	SL -			
	N Set if the most significant bit of the result is set. Cleared otherwise.									
	Z	Set if the result equals zero. Cleared otherwise.								
	С	Set if a borrow is generated. Cleared otherwise.								
	SV	Not affected.								
	E	Not a	affected.							
Addrossing Medes	SL	Not affected.				D	<b>F</b>			
Addressing wodes		Rw. Rw				кер No	FOI A3	nm C2 00	Bytes 4	
	CoCMP	[IDX:⊗] [Rw <sub>∞</sub> ⊗]				No	93	Xm C2 0:0aaa	4	
	CoCMP	Rw <sub>n</sub> , [Rw <sub>m</sub> ⊗]				No	83	nm C2 0:0qqq	4	
Examples	CoCMP	[IDX1+QX0], [R11+QR1] R1, [R2-]				; MSW(N,Z,C) $\leftarrow$ (ACC) - ((R11))\((IDX1)) ; (R11) $\leftarrow$ (R11) + (QR1) ; (IDX1) $\leftarrow$ (IDX1) + (QX0) ; MSW(N,Z,C) $\leftarrow$ (ACC) - ((R2))\(R1) ; (R2) $\leftarrow$ (R2) - 2				
	CoCMP									
	CoCMP	R2, R5				; MSW(N,Z,C) ← (ACC) - (R5)\(R2)				

CoSTORE	Store a MAC-Unit Register								
Group	Transfer Instructions								
Syntax	CoSTOR	E o	op1, op2						
Operation	(op1) ← (op2)								
Data Types	WORD								
Description	Moves the contents of a MAC-Unit register specified by the source operand op2 to the location specified by the destination operand op1. This instruction is repeatable with destination indirect addressing mode (for example to clear a table in memory)								
MAC Condition									
Flags	N	Z	С	SV	E		SL		
	-	-	-	-	-		-		
	Ν	Not	affected						
	Z	Not a	affected						
	С	C Not affected							
	SV	V Not affected							
	E Not affected								
	SL	SL Not affected							
Addressing Modes	Mnemon	ic			Rep	Forma	at		Bytes
	CoSTOR	E Rv	Rw <sub>n</sub> , CoReg			C3 nn www:w000 00 4			4
	CoSTOR	E [R	w <sub>n</sub> ⊕], Co	Reg	Yes	B3 nn	www	pppr:rrr 000w:w	4
Note	Due to pipeline side effects, CoSTORE cannot be directly followed by a MOV instruction the source operand of which is also a MAC-Unit Register like MSW, MAH, MAL, MAS(u), MRW, or MCW. In that particular case a NOP must be inserted between the CoSTORE and the MOV instruction.								
Examples	CoSTORE	E [R11-	+QR1], MA	AS		; ((R11 ; (R11)	I )) ← li ) ←(R′	mited((ACC)) I1) + (QR1)	
	Repeat 3 times CoSTORE [R2-], MAL; ((R2)) $\leftarrow$ (MAL); (R2) $\leftarrow$ (R2) - 2								

57

## CoMOV

CoMOV	Memory to Memory Move									
Group	Transfer Instructions									
Syntax	CoMOV op1, op2									
Operation	(op1) ←	(op1) ← (op2)								
Data Types	WORD									
Description	Moves the contents of the memory location specified by the source operand, op2, to the memory location specified by the destination operand op1. This instruction is repeatable. Note that, unlike for the other instructions, IDXi can address the entire memory. This instruction does not affect the Mac Condition Flags but modify the CPU Condition Flags as any other MOV instruction.									
CPU Condition	_	-	v	•						
Flags	E *	<b>ک</b> *	- V	-	N *					
	E Set if the value of op2 represents the lowest possible negative									
	Z Set if the value of the source operand op2 equals zero. Cleared otherwise.									
	V	V Not affected.								
	C Not affected.									
	N Set if the most significant bit of the source operand op2 is set Cleared otherwise									
MAC Condition										
Flags	N	Z	С	SV	Е	SL	1			
	-	-	-	-	-	-				
	Ν	Not a	ffected.							
	Z	Not a	Not affected.							
	С	Not a	Not affected.							
	SV	Not a	iffected.							
	E	Not a	iffected.							
Addrossing Modes	SL	NOT a	mected.		Dee	Γ.		Dutes		
Addressing modes	CoMOV	IIC [ID]	X <sub>i</sub> ⊗], [Rv	/ <sub>m</sub> ⊗]	Yes	D3	3 Xm 00 rrrr:rqqq	4		
Examples	Repeat 24 times CoMOV [IDX1+QX0], [R11+QR ; (( ; (I ; (I					1] IDX1)) ← R11) ← (F DX1) ← (	- ((R11)) R11) + (QR1) IDX1) + (QX0)			

**57** 

Information furnished is believed to be accurate and reliable. However, SGS-THOMSON Microelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of SGS-THOMSON Microelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. SGS-THOMSON Microelectronics products are not authorized for use as critical components in life support devices or systems without the express written approval of SGS-THOMSON Microelectronics.

©1997 SGS-THOMSON Microelectronics - All rights reserved.

SGS-THOMSON Microelectronics Group of Companies Australia - Brazil - Canada - China - France - Germany - Hong Kong - Italy - Japan - Korea - Malaysia - Malta - Morocco The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - Thailand - United Kingdom - U.S.A.

