ST10 FAMILY PROGRAMMING MANUAL

2nd EDITION
AUGUST 1997

SGS-THOMSON MICROELECTRONICS

Table of Contents

1 INTRODUCTION 6
STANDARD INSTRUCTION SET
2 ADDRESSING MODES 7
2.1 Short addressing modes 7
2.2 Long addressing mode 9
2.3 Indirect addressing modes 11
2.4 Constants 12
2.5 Branch target addressing modes 13
3 INSTRUCTION EXECUTION TIMES 14
3.1 Definition of measurement units 14
3.2 Minimum state times 15
3.3 Additional state times 16
4 INSTRUCTION SET SUMMARY 19
4.1 X-ref tables of opcode by mnemonic 19
4.2 X-ref table of mnemonic, address mode & number of bytes 20
4.3 Instruction set ordered by functional group 22
4.4 Instruction set ordered by opcodes 37
4.5 Instruction conventions 43
4.6 Notes on ATOMIC and EXTended instructions 48
5 INDIVIDUAL INSTRUCTION DESCRIPTIONS 48
ADD 49
ADDB 50
ADDC 51
ADDBC 52
AND 53
ANDB 54
ASHR 55
ATOMIC 56
BAND 57
BCLR 58
BCMP 59
BFLDH 60
BFLDL 61
BMOV 62
BMOVN 63

Table of Contents

BOR 64
BSET 65
BXOR 66
CALLA 67
CALLI 68
CALLR 69
CALLS 70
CMP 71
CMPB 72
CMPD1 73
CMPD2 74
CMPI1 75
CMPI2 76
CPL 77
CPLB 78
DISWDT 79
DIV 80
DIVL 81
DIVLU 82
DIVU 83
EINIT 84
EXTR 85
EXTP 86
EXTPR 87
EXTS 88
EXTSR 89
IDLE 90
JB 91
JBC 92
JMPA 93
JMPI 94
JMPR 95
JMPS 96
JNB 97
JNBS 98

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 133
7 MAC INSTRUCTION EXECUTION TIME 134
8 MAC INSTRUCTION SET SUMMARY 135
8.1 MAC instruction conventions 137
9 INDIVIDUAL INSTRUCTION DESCRIPTION 139
CoMUL(-) 140
CoMULu(-) 142
CoMULsu(-) 144
CoMULus(-) 146
CoMAC(R/-) 148
CoMAC(R)u(-) 151
CoMAC(R)su(-) 153
CoMAC(R)us(-) 155
CoMACM(R/-) 157
CoMACM(R)u(-) 160
CoMACM(R)su(-) 162
CoMACM(R)us(-) 164
CoADD(2) 166
CoSUB(2)(R) 168
CoNEG 170
CoABS 171
CoLOAD(2)(-) 172
CoNOP 174
CoSHL 175
CoSHR 176
CoASHR 177
CoRND 179
CoMAX 180
CoMIN 181
CoCMP 182
CoSTORE 183
CoMOV 184

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.

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 + Δ * Short Address

Note: Δ is 1 for byte GPRs, Δ is 2 for word GPRs.

Table 2.1 Short addressing mode summary

Mnemonic	Physical Address	Short Address Range	Scope of Access
Rw	(CP) + 2*Rw	Rw = 0...15	GPRs (Word) 16 values
Rb	(CP) + 1*Rb	Rb = 0...15	GPRs (Byte) 16 values
reg	00’FE00h + 2*reg 00’F000h + 2*reg *) (CP) + 2*(reg∧0Fh) (CP) + 1*(reg∧0Fh)	reg = 00h...EFh reg = 00h...EFh reg = F0h...FFh reg = F0h...FFh	SFRs (Word, Low byte) ESFRs (Word, Low byte)*) GPRs (Word) 16 values GPRs (Bytes) 16 values
bitoff	00’FD00h + 2*bitoff 00’FF00h + 2*(bitoff∧FFh) (CP) + 2*(bitoff∧0Fh)	bitoff = 00h...7Fh bitoff = 80h...EFh bitoff = F0h...FFh	RAM Bit word offset 128 values SFR Bit word offset 128 values GPR Bit word offset 16 values
bitaddr	Word offset as with bitoff. Immediate bit position.	bitoff = 00h...FFh bitpos = 0...15	Any single bit

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

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’FFDEh) or use 00’F100h as a base address to specify the highest internal ESFR word locations (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.

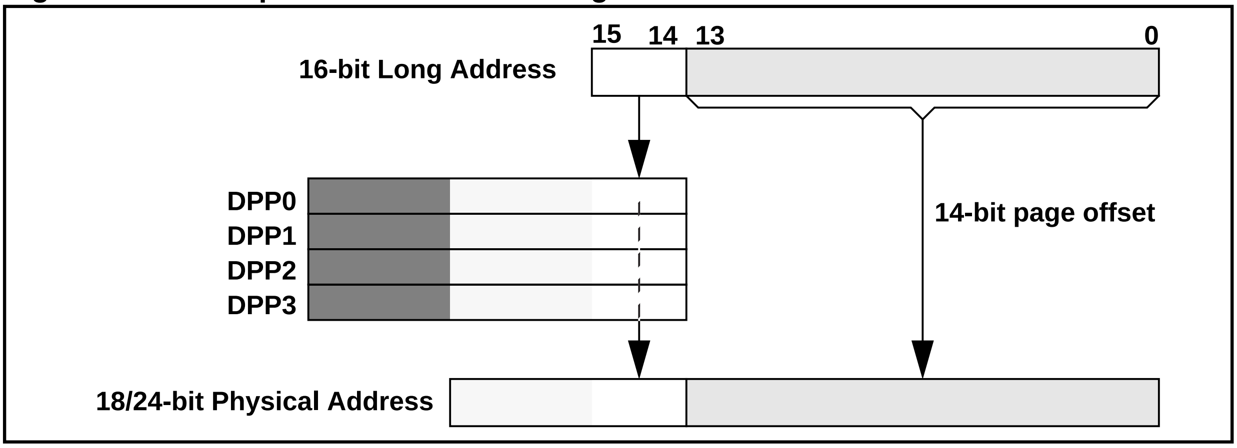
2.2 Long addressing mode

Long addressing mode uses one of the four DPP registers to specify a physical 18-bit or 24-bit 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 address.

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

Figure 2.2 Summary of long address modes

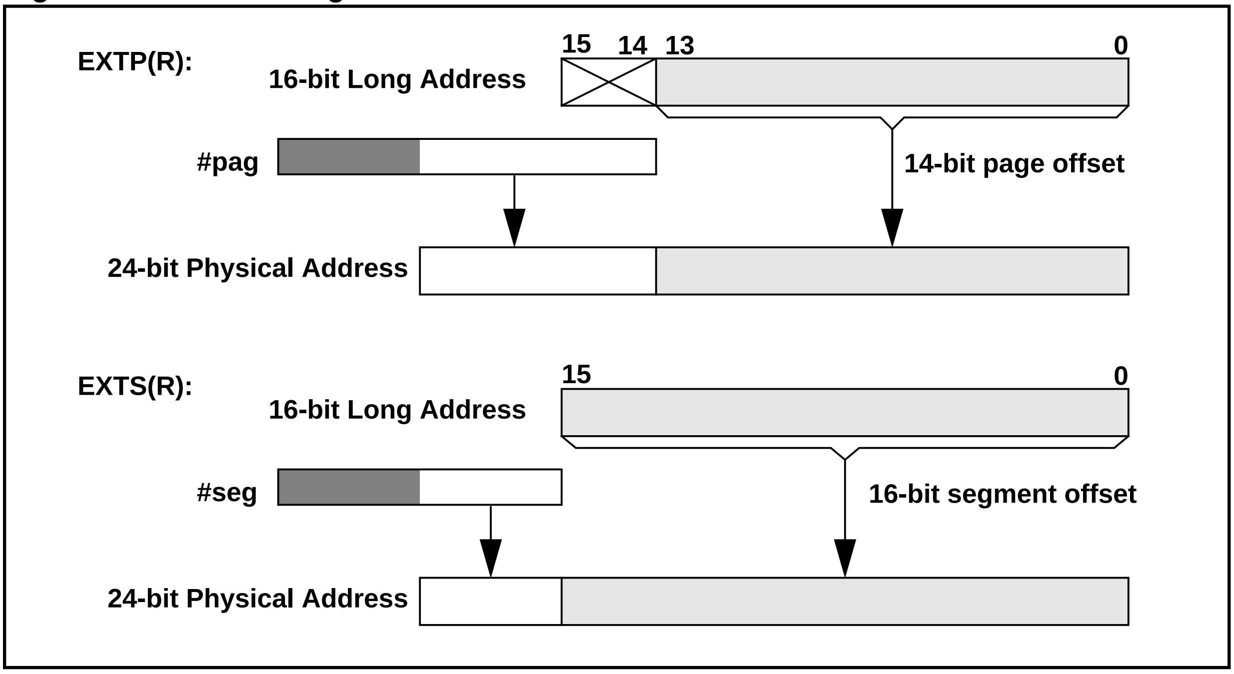
Mnemonic	Physical Address	Long Address Range	Scope of Access
mem	(DPP0) || mem$\wedge$3FFFh (DPP1) || mem$\wedge$3FFFh (DPP2) || mem$\wedge$3FFFh (DPP3) || mem$\wedge$3FFFh	0000h...3FFFh 4000h...7FFFh 8000h...BFFFh C000h...FFFFh	Any Word or Byte
mem	pag || mem$\wedge$3FFFh	0000h...FFFFh (14-bit)	Any Word or Byte
mem	seg || mem	0000h...FFFFh (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 - Δ; [optional step!]

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

(GPR Address) = (GPR Address) - Δ; [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 data-type-dependent value (Δ=1 for byte operations, Δ=2 for word operations):

(GPR Pointer) = (GPR Pointer) + Δ; [optional step!]

The following indirect addressing modes are provided:

Table 2.2 Table of indirect address modes

Mnemonic	Notes
[Rw]	Most instructions accept any GPR (R15...R0) as indirect address pointer. Some instructions, however, only accept the lower four GPRs (R3...R0).
[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

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 2.3 Table of constants

Mnemonic	Word Operation	Byte Operation
#data3	0000h + data3	00h + data3
#data4	0000h + data4	00h + data4
#data8	0000h + data8	data8
#data16	data16	data16 $\wedge$ 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.

Table 2.4 Branch target address summary

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000h...FFFEh
rel	(IP) = (IP) + 2*rel (IP) = (IP) + 2*(~rel+1)	- -	rel = 00h...7Fh rel = 80h...FFh
[Rw]	(IP) = ((CP) + 2*Rw)	-	Rw = 0...15
seg	-	(CSP) = seg	seg = 0...3
#trap7	(IP) = 0000h + 4*trap7	(CSP) = 0000h	trap7 = 00h...7Fh

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_h) 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_{CPU}]: 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 f_{CPU} = variable
	= 50	[ns]	; for f_{CPU} = 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 ns ... 900 ns ; for f_{CPU} = 20 MHz

For multiplexed external bus modes:

1*ACT = 3 + (15 – MCTC) + (1 – MTTC) * States

= 150 ns ... 950 ns ; for f_{CPU} = 20 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_{In}) of the considered instructions plus an offset value of 6 state times which takes into account the solitary filling of the pipeline, as follows:

$T_{tot} = T_{I1} + T_{I2} + ... + T_{In} + 6 * \text{States}$

T_{In} 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_{Imin} (ROM)). This table can also be used to calculate the minimum number of state times for instructions fetched from the internal RAM (T_{Imin} (RAM)), or ALE Cycle Times for instructions fetched from the external memory (T_{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_{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:

Table 3.1 Minimum instruction state times [Unit = ns]

Instruction	T_{Imin} (ROM) [States]		T_{Imin} (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	

For 2-byte instructions: $T_{Imin}(RAM) = T_{Imin}(ROM) + 4 * States$

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

Unlike internal ROM program execution, the minimum time $T_{Imin}(ext)$ to process an external instruction also depends on instruction length. $T_{Imin}(ext)$ 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: $T_{Imin}(ext) = 1*ACT + (T_{Imin}(ROM) - 2) * States$

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

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: $T_{Iadd} = 2 * States$

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

Internal RAM operand reads via indirect addressing modes: T_{Iadd} = 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 auto-increments or auto-decrements a GPR, as shown in the following example:

I_n	: MOV R1, [R0+]	; auto-increment R0
I_{n+1}	: MOV [R3], [R2]	; if R2 points into the internal RAM space: ; T_{Iadd} = 1 * 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_{Iadd} = 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:

I_n	: MOV T0, #1000h	; write to Timer 0
I_{n+1}	: ADD R3, T1	; read from Timer 1: T_{Iadd} = 1 * State

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

I_n	: ADD R0, #1000h	; implicit modification of PSW flags
I_{n+1}	: BAND C, Z	; read from PSW: T_{Iadd} = 2 * 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_n	: MOV SP, #0FB00h	; explicit update of the stack pointer
I_{n+1}	: SCX R1, #1000h	; implicit decrement of the stack pointer:
		: T_{Iadd} = 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_{Iadd} = 1 * ACT

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_{Iadd} = 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_{Iadd} 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 are normally performed in parallel to other CPU operations. Thus, T_{Iadd} 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: T_{Iadd} = 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_{n+1}	: JMPA cc–UC, label	; if a standard branch is taken: : T_{Iadd} = 2 * States (T_{In} = 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_{t+1}	:	; any non-aligned double word instruction : (e.g. at location 12FEh)
I_{n+1}	:JMPR cc–UC, label	; provided that a cache jump is taken: : T_{Iadd} = 2 * States (T_{In} = 4 * 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_{Iadd} = 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_n	: BSET USR0	; write to PSW
I_{n+1}	:JMPR cc–Z, label	;test condition flag in PSW: T_{Iadd}= 1 * State

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

Table 4.1 Hexadecimal 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
xB	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
xE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

Table 4.2 Hexadecimal opcode by mnemonic

	8x	9x	Ax	Bx	Cx	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
xB	-	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
xE	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).

Table 4.3 Instruction mnemonic by address modes and number of bytes

Mnemonic	Addressing Modes	Bytes		Mnemonic	Addressing Modes	Bytes
ADD[B]	Rwn Rwm [1]	2		CPL[B]	Rwn [1]	2
ADDC[B]	Rwn Rwi] [1]	2		NEG[B]		
AND[B]	Rwn Rwi+] [1]	2		DIV	Rwn	2
OR[B]	Rwn #data3 [1]	2		DIVL		
SUB[B]	reg #data16 [2]	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 [1]	2
BCMP					Rwn [Rwi] [1]	2
BMOV					Rwn [Rwi+] [1]	2
BMOVN					Rwn #data3 [1]	2
BOR /					reg #data16 [2]	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 address modes and number of bytes (cont'd)

Mnemonic	Addressing Modes	Bytes
MOV[B]	Rwn Rwm [1] Rwn #data4 [1] Rwn Rwm] [1] Rwn Rwm+] [1] [Rwm Rwn [1] [-Rwm] Rwn [1] [Rwn] [Rwm] [Rwn+] [Rwm] [Rwn] [Rwm+] reg #data16 [2] Rwn [Rwm+#d16] [1] [Rwm+#d16] Rwn [1] [Rwn] mem mem [Rwn] reg mem mem reg	2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4
MOVBS MOVBZ	Rwn Rbm reg mem mem reg	2 4 4
EXTS EXTSR	Rwm #data2 [3] #seg #data2	2 4
NOP RET RETI RETS	-	2

Mnemonic	Addressing Modes	Bytes
CALLS JMPS	segcaddr	4
CALLR	rel	2
JMPR	cc rel	2
JB JBC JNB JNBS	bitaddrQ.q rel	4
PCALL	reg caddr	4
POP PUSH RETP	reg	2
SCXT	reg #data16 reg mem	4 4
PRIOR	Rwn Rwm	2
TRAP	#trap7	2
ATOMIC EXTR	#data2 [3]	2
EXTP EXTPR	Rwm #data2 [3] #pag #data2	2 4
SRST/IDLE PWRDN SRVWDT DISWDT EINIT	-	4

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:) 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 and post- increment source pointer by 2	2	6	2	3	4	6	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
ADDB Rb, [Rw +]	Add indirect byte memory to direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
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
ADDC Rw, Rw	Add direct word GPR to direct GPR with Carry	2	6	2	3	4	6	2
ADDC Rw, [Rw]	Add indirect word memory to direct GPR with Carry	2	6	2	3	4	6	2
ADDC Rw, [Rw +]	Add indirect word memory to direct GPR with Carry and post-increment source pointer by 2	2	6	2	3	4	6	2
ADDC Rw, #data3	Add immediate word data to direct GPR with Carry	2	6	2	3	4	6	2
ADDC reg, #data16	Add immediate word data to direct register with Carry	2	8	4	6	8	12	4
ADDC reg, mem	Add direct word memory to direct register with Carry	2	8	4	6	8	12	4
ADDC mem, reg	Add direct word register to direct memory with Carry	2	8	4	6	8	12	4
ADDCB Rb, Rb	Add direct byte GPR to direct GPR with Carry	2	6	2	3	4	6	2
ADDCB Rb, [Rw]	Add indirect byte memory to direct GPR with Carry	2	6	2	3	4	6	2
ADDCB Rb, [Rw +]	Add indirect byte memory to direct GPR with Carry and post-increment source pointer by 1	2	6	2	3	4	6	2

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
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
MULU Rw, 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
NEGB Rb	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)

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 Arithmetic 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 register with Carry	2	8	4	6	8	12	4
SUBCBmem, reg	Subtract direct byte register from direct memory with Carry	2	8	4	6	8	12	4

Table 4.5 Logical 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 direct register	2	8	4	6	8	12	4
AND mem, reg	Bitwise AND direct word register with direct 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 direct register	2	8	4	6	8	12	4
ANDB mem, reg	Bitwise AND direct byte register with direct memory	2	8	4	6	8	12	4

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
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 direct register	2	8	4	6	8	12	4
OR	reg, mem	Bitwise OR direct word memory with direct register	2	8	4	6	8	12	4
OR	mem, reg	Bitwise OR direct word register with direct 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 direct GPR	2	6	2	3	4	6	2
ORB	Rb, [Rw +]	Bitwise OR indirect byte memory with direct 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 direct register	2	8	4	6	8	12	4
ORB	reg, mem	Bitwise OR direct byte memory with direct register	2	8	4	6	8	12	4
ORB	mem, reg	Bitwise OR direct byte register with direct 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 direct memory	2	8	4	6	8	12	4
XORB Rb, Rb	Bitwise XOR direct byte GPR with direct GPR	2	6	2	3	4	6	2
XORB Rb, [Rw]	Bitwise XOR indirect byte memory with direct GPR	2	6	2	3	4	6	2
XORB Rb, [Rw +]	Bitwise XOR indirect byte memory with direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
XORB Rb, #data3	Bitwise XOR immediate byte data with direct GPR	2	6	2	3	4	6	2
XORB reg, #data16	Bitwise XOR immediate byte data with direct register	2	8	4	6	8	12	4
XORB reg, mem	Bitwise XOR direct byte memory with direct register	2	8	4	6	8	12	4
XORB mem, reg	Bitwise XOR direct byte register with direct memory	2	8	4	6	8	12	4

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 bitaddr, bitaddr	AND direct bit with direct bit	2	8	4	6	8	12	4
BCLR bitaddr	Clear direct bit	2	6	2	3	4	6	2
BCMP bitaddr, bitaddr	Compare direct bit to direct bit	2	8	4	6	8	12	4
BFLDH bitoff, #mask8,#data8	Bitwise modify masked high byte of bit-addressable direct word memory with immediate data	2	8	4	6	8	12	4
BFLDL bitoff, #mask8, #data8	Bitwise modify masked low byte of bit-addressable direct word memory with immediate data	2	8	4	6	8	12	4
BMOV bitaddr, bitaddr	Move direct bit to direct bit	2	8	4	6	8	12	4

Table 4.6 Boolean 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
BMOVN bitaddr, bitaddr	Move negated direct bit to direct bit	2	8	4	6	8	12	4
BOR 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 bitaddr, bitaddr	XOR direct bit with direct bit	2	8	4	6	8	12	4
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 4.7 Compare and loop instructions

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 4.8 Prioritize instructions

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 normalize direct word GPR and store result in direct word GPR	2	6	2	3	4	6	2

Table 4.9 Shift and rotate instructions

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 4.10 Data 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

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
MOV [Rw +], [Rw]	Move indirect word memory to indirect memory and post-increment destination pointer by 2	2	6	2	3	4	6	2
MOV [Rw], [Rw +]	Move indirect word memory to indirect memory 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 constant 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
MOVB Rb, Rb	Move direct byte GPR to direct GPR	2	6	2	3	4	6	2
MOVB Rb, #data4	Move immediate byte data to direct GPR	2	6	2	3	4	6	2
MOVB reg, #data16	Move immediate byte data to direct register	2	8	4	6	8	12	4
MOVB Rb, [Rw]	Move indirect byte memory to direct GPR	2	6	2	3	4	6	2
MOVB Rb, [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 constant to direct GPR	4	10	6	8	10	14	4
MOVB [Rw + #data16], Rb	Move direct byte GPR to indirect memory by base plus constant	2	8	4	6	8	12	4
MOVB [Rw], mem	Move direct byte memory to indirect memory	2	8	4	6	8	12	4
MOVB mem, [Rw]	Move indirect byte memory to direct memory	2	8	4	6	8	12	4
MOVB reg, mem	Move direct byte memory to direct register	2	8	4	6	8	12	4
MOVB mem, reg	Move direct byte register to direct memory	2	8	4	6	8	12	4
MOVBS Rw, Rb	Move direct byte GPR with sign extension to direct word GPR	2	6	2	3	4	6	2
MOVBS reg, 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)

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/ 8	6/4	8/6	10/ 8	14/ 12	4
CALLI cc, [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 segment	4	10	6	8	10	14	4
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/ 8	6/4	8/6	10/ 8	14/ 12	4
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 set	4	10	6	8	10	14	4
PCALLreg, caddr	Push direct word register onto system stack and call absolute subroutine	4	10	6	8	10	14	4
TRAP #trap7	Call interrupt service routine via immediate trap number	4	8	4	5	6	8	2

Table 4.12 System Stack Instructions

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 immediate 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 4.13 Return 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 system 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*)	2	6	2	3	4	6	2
EXTP #pag10, #data2	Begin EXTended Page sequence*)	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*)	2	6	2	3	4	6	2

Table 4.14 System Control Instructions (cont'd)

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

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

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

Table 4.16 Instruction set ordered by Hex code

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 [1)]
09	2	ADDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
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 [1)]
19	2	ADDCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
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 (cont'd)

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 [1)]
29	2	SUBB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
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 [1)]
39	2	SUBCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
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	CMPB	Rb, Rb
42	4	CMP	reg, mem
43	4	CMPB	reg, mem
44	-	-	-
45	-	-	-
46	4	CMP	reg, #data16
47	4	CMPB	reg, #data8
48	2	CMP	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
49	2	CMPB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 [1)]
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)

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 1)
59	2	XORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 1)
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 1)
69	2	ANDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 1)
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 1)
79	2	ORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3 1)
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)

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

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)
	←	(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)
	∨	(opX)	is	logically ORed with (opY)
	⊕	(opX)	is	logically EXCLUSIVELY ORed with (opY)
	⇔	(opX)	is	COMPARED against (opY)
	mod	(opX)	is	divided MODULO (opY)
Monadic operations		(opY)		operator (opX)
	¬	(opX)	is	logically COMPLEMENTED

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

opX	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX_n)	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:

CP	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
C	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.

Table 4.17 Condition codes

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
cc_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	$(Z \vee C) = 1$	Unsigned less than or equal	Fh
cc_UGE	C = 0	Unsigned greater than or equal	9h
cc_UGT	$(Z \vee C) = 0$	Unsigned greater than	Eh
cc_SLT	$(N \oplus V) = 1$	Signed less than	Ch
cc_SLE	$(Z \vee (N \oplus V)) = 1$	Signed less than or equal	Bh
cc_SGE	$(N \oplus V) = 0$	Signed greater than or equal	Dh
cc_SGT	$(Z \vee (N \oplus V)) = 0$	Signed greater than	Ah
cc_NET	$(Z \vee E) = 0$	Not equal AND not end of table	1h

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

Table 4.18 List of condition flags

Symbol	Description
*	The flag is set according to the following standard rules
	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 is set according to non-standard rules. Individual instruction pages or the ALU status flags description.
'-'	The flag is not affected by the operation
'0'	The flag is cleared by the operation.
'NOR'	The flag 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 contains the original value of the specified bit operand.
'$\overline{B}$'	The flag contains the complemented value of the specified bit operand

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.

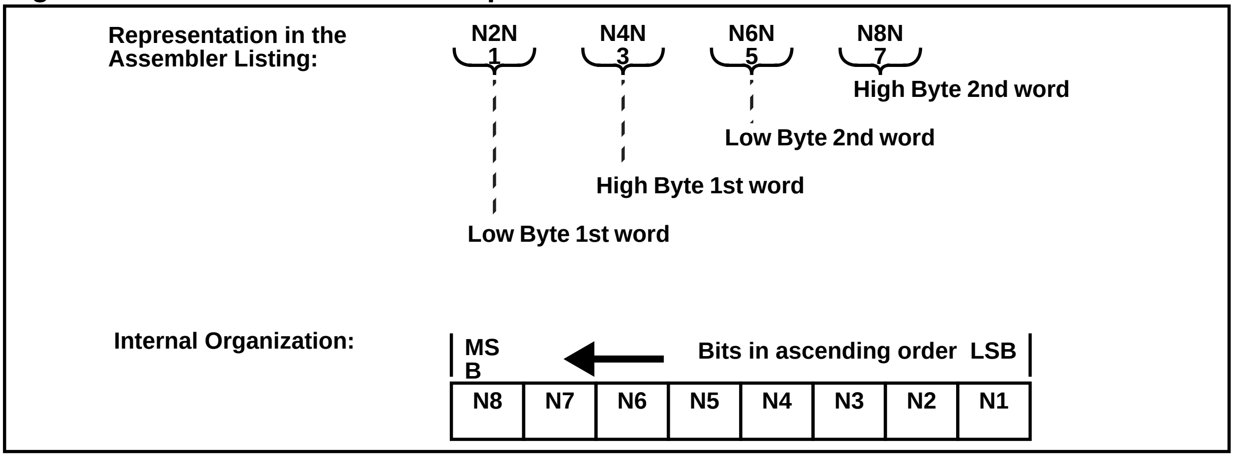
- **Mnemonic** gives an example of which operands the instruction will accept.
- **Format** specifies the format of the instruction as used in the assembler listing. Figure 4.1 shows the reference between the instruction format representation of the assembler and the corresponding internal organization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

Table 4.19 Instruction format symbols

00_h through FF_h	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)
c	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)
pp 0:00pp	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)

- **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.

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) ← (op1) + (op2)

Data Types

WORD

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 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 overflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADD	Rw_n, Rw_m	00 nm	2
ADD	Rw_n, [Rw_i]	08 n:10ii	2
ADD	Rw_n, [Rw_i+]	08 n:11ii	2
ADD	Rw_n, #$data_3$	08 n:0###	2
ADD	reg, #$data_{16}$	06 RR ## ##	4
ADD	reg, mem	02 RR MM MM	4
ADD	mem, reg	04 RR MM MM	4

ADDB

Integer Addition

Syntax

ADDB op1, op2

Operation

(op1) ← (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 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 overflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.
C	Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDB	Rb_n, Rb_m	01 nm	2
ADDB	Rb_n, [Rw_i]	09 n:10ii	2
ADDB	Rb_n, [Rw_i+]	09 n:11ii	2
ADDB	Rb_n, #$data_3$	09 n:0###	2
ADDB	reg, #$data_{16}$	07 RR ## ##	4
ADDB	reg, mem	03 RR MM MM	4
ADDB	mem, reg	05 RR MM MM	4

ADDC

Integer Addition with Carry

Syntax

ADDC op1, op2

Operation

(op1) ← (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	Z	V	C	N
*	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 and previous Z flag was set. Cleared otherwise.

V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDC	Rw_n, Rw_m	10 nm	2
ADDC	Rw_n, [Rw_i]	18 n:10ii	2
ADDC	Rw_n, [Rw_i+]	18 n:11ii	2
ADDC	Rw_n, #$data_3$	18 n:0###	2
ADDC	reg, #$data_{16}$	16 RR ## ##	4
ADDC	reg, mem	12 RR MM MM	4
ADDC	mem, reg	14 RR MM MM	4

ADDBC

Integer Addition with Carry

Syntax

ADDBC op1, op2

Operation

(op1) ← (op1) + (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

E	Z	V	C	N
*	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 and previous Z flag was set. Cleared otherwise.

V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a carry is generated from the most significant bit of the specified data type. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ADDCB	Rb_n, Rb_m	11 nm	2
ADDCB	Rb_n, [Rw_i]	19 n:10ii	2
ADDCB	Rb_n, [Rw_i+]	19 n:11ii	2
ADDCB	Rb_n, #$data_3$	19 n:0###	2
ADDCB	reg, #$data_{16}$	17 RR ## ##	4
ADDCB	reg, mem	13 RR MM MM	4
ADDCB	mem, reg	15 RR MM MM	4

AND

Logical AND

Syntax

AND op1, op2

Operation

(op1) $\leftarrow$ (op1) $\wedge$ (op2)

Data Types

WORD

Description

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

E	Z	V	C	N
*	*	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.
V	Always cleared.
C	Always cleared.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
AND	Rw_n, Rw_m	60 nm	2
AND	Rw_n, [Rw_i]	68 n:10ii	2
AND	Rw_n, [Rw_i+]	68 n:11ii	2
AND	Rw_n, #$data_3$	68 n:0###	2
AND	reg, #$data_{16}$	66 RR ## ##	4
AND	reg, mem	62 RR MM MM	4
AND	mem, reg	64 RR MM MM	4

ANDB

Logical AND

Syntax

ANDB op1, op2

Operation

(op1) ← (op1) ∧ (op2)

Data Types

BYTE

Description

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

E	Z	V	C	N
*	*	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.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ANDB	Rb_n, Rb_m	61 nm	2
ANDB	Rb_n, [Rw_i]	69 n:10ii	2
ANDB	Rb_n, [Rw_i+]	69 n:11ii	2
ANDB	Rb_n, #$data_3$	69 n:0###	2
ANDB	reg, #$data_{16}$	67 RR ## ##	4
ANDB	reg, mem	63 RR MM MM	4
ANDB	mem, reg	65 RR MM MM	4

ASHR

Arithmetic Shift Right

Syntax

ASHR op1, op2

Operation

(count) $\leftarrow$ (op1) $\wedge$ (op2)
(V) $\leftarrow$ 0
(C) $\leftarrow$ 0
DO WHILE (count) $\neq$ 0
 (V) $\leftarrow$ (C) $\vee$ (V)
 (C) $\leftarrow$ ($op1_0$)
 ($op1_n$) $\leftarrow$ ($op1_{n+1}$) [n=0...14]
 (count) $\leftarrow$ (count) - 1
END WHILE

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 significant 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	Z	V	C	N
0	*	S	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V 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 shifted out of op1. Cleared for a shift count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ASHR	Rw_n, Rw_m	AC nm	2
ASHR	Rw_n, $\#data_4$	BC #n	2

ATOMIC

Begin ATOMIC Sequence

Syntax

ATOMIC op1

Operation

(count) $\leftarrow$ (op1) [$1 \leq$ op1 ≤ 4]
Disable interrupts and Class A traps
DO WHILE ((count) $\neq$ 0 AND Class_B_trap_condition $\neq$ TRUE)
 Next Instruction
 (count) $\leftarrow$ (count) - 1
END WHILE
(count) = 0
Enable 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 ATOMIC instruction must be used carefully (see Section 4.6 “Notes on ATOMIC and EXTended instructions”).
The ATOMIC instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic	Format	Bytes
ATOMIC #$data_2$	D1 :00##-0	2

BAND

Bit Logical AND

Syntax

BAND op1, op2

Operation

(op1) ← (op1) ∧ (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	Z	V	C	N
0	NOR	OR	AND	XOR

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

Mnemonic	Format	Bytes
BAND $bitaddr_{Z.z}$, $bitaddr_{Q.q}$	6A QQ ZZ qz	4

BCLR

Bit Clear

Syntax

BCLR op1

Operation

(op1) ← 0

Data Types

BIT

Description

CLears the bit specified by op1. This instruction is primarily used for peripheral and system control.

Condition Flags

E	Z	V	C	N
0	B	0	0	B

E Always cleared.

Z Contains the logical negation of the previous state of the specified bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the specified bit.

Addressing Modes

Mnemonic		Format	Bytes
BCLR	$bitaddr_{Q.}$	qE QQ	2

BCMP

Bit to Bit Compare

Syntax

BCMP op1, op2

Operation

(op1) ⇔ (op2)

Data Types

BIT

Description

Performs a single bit comparison of the source bit specified by operand op1 to the source bit specified by operand op2. No result is written by this instruction. Only the condition codes are updated.

Condition Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

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

Mnemonic		Format	Bytes
BCMP	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	2A QQ ZZ qz	4

BFLDH

Bit Field High Byte

Syntax

BFLDH op1, op2, op3

Operation

(tmp) $\leftarrow$ (op1)
(high byte (tmp)) $\leftarrow$ ((high byte (tmp) $\wedge$ $\neg$op2) $\vee$ op3)
(op1) $\leftarrow$ (tmp)

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

E	Z	V	C	N
0	*	0	0	*

E Always cleared.

Z Set if the word result equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the word result is set. Cleared otherwise.

Addressing Modes

Mnemonic	Format	Bytes
BFLDH $bitoff_Q$, $\#mask_8$, $\#data_8$	1A QQ ## @@	4

BFLDL

Bit Field Low Byte

Syntax

BFLDL op1, op2, op3

Operation

(tmp) $\leftarrow$ (op1)
(low byte (tmp)) $\leftarrow$ ((low byte (tmp) $\wedge \neg$op2) $\vee$ op3)
(op1) $\leftarrow$ (tmp)

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

E	Z	V	C	N
0	*	0	0	*

E Always cleared.

Z Set if the word result equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the word result is set. Cleared otherwise.

Addressing Modes

Mnemonic	Format	Bytes
BFLDL bitoff$_Q$, #mask$_8$, #data$_8$	0A QQ @@##	4

BMOV

Bit to Bit Move

Syntax

BMOV op1, op2

Operation

(op1) $\leftarrow$ (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

Condition Flags	E	Z	V	C	N
	0	B	0	0	B

E Always cleared.

Z Contains the logical negation of the previous state of the source bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the source bit.

Addressing Modes

Mnemonic	Format	Bytes
BMOV bitaddr$_{Z.z}$, bitaddr$_{Q.q}$	4A QQ ZZ qz	4

BMOVN

Bit to Bit Move and Negate

Syntax

BMOVN op1, op2

Operation

(op1) $\leftarrow \neg$(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

E	Z	V	C	N
0	B	0	0	B

E Always cleared.

Z Contains the logical negation of the previous state of the source bit.

V Always cleared.

C Always cleared.

N Contains the previous state of the source bit.

Addressing Modes

Mnemonic	Format	Bytes
BMOVN bitaddr$_{Z.z}$, bitaddr$_{Q.q}$	3A QQ ZZ qz	4

BOR

Bit Logical OR

Syntax

BOR op1, op2

Operation

(op1) $\leftarrow$ (op1) $\vee$ (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

E	Z	V	C	N
0	NOR	OR	AND	XOR

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

Mnemonic	Format	Bytes
BOR bitaddr$_{Z.z}$, bitaddr$_{Q.q}$	5A QQ ZZ qz	4

BSET

Bit Set

Syntax

BSET op1

Operation

(op1) $\leftarrow$ 1

Data Types

BIT

Description

Sets the bit specified by op1. This instruction is primarily used for peripheral and system control.

Condition Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E	Always cleared.
Z	Contains the logical negation of the previous state of the specified bit.
V	Always cleared.
C	Always cleared.
N	Contains the previous state of the specified bit.

Addressing Modes

Mnemonic		Format	Bytes
BSET	$\text{bitaddr}_{Q.q}$	qF QQ	2

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

E	Z	V	C	N
0	NOR	OR	AND	XOR

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

Mnemonic	Format	Bytes
BXOR bitaddr$_{Z.z}$, bitaddr$_{Q.q}$	7A QQ ZZ qz	4

CALLA

Call Subroutine Absolute

Syntax

CALLA op1, op2

Operation

IF (op1) THEN
(SP) $\leftarrow$ (SP) - 2
((SP)) $\leftarrow$ (IP)
(IP) $\leftarrow$ op2
ELSE
next instruction
END IF

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.

Condition Codes

See condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLA	cc, caddr	CA c0 MM MM	4

CALLI

Call Subroutine Indirect

Syntax

CALLI op1, op2

Operation

IF (op1) THEN
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← (op2)
ELSE
next instruction
END IF

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 condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
CALLI	cc, [Rw_n]	AB cn2

CALLR

Call Subroutine Relative

Syntax

CALLR op1

Operation

(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← (IP) + sign_extend (op1)

Description

A branch is taken to the location specified by the instruction pointer, IP, plus the relative displacement, op1. The displacement 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 represents 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 condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLR	rel	BB rr	2

CALLS

Call Inter-Segment Subroutine

Syntax

CALLS op1, op2

Operation

(SP) ← (SP) - 2
((SP)) ← (CSP)
(SP) ← (SP) - 2
((SP)) ← (IP)
(CSP) ← op1
(IP) ← op2

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 condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic		Format	Bytes
CALLS	seg, caddr	DA ss MM MM	4

CMP

Integer Compare

Syntax

CMP op1, op2

Operation

(op1) ⇔ (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

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.
C	Set if a borrow is generated. Cleared otherwise.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMP	Rw_n, Rw_m	40 nm	2
CMP	Rw_n, [Rw_i]	48 n:10ii	2
CMP	Rw_n, [Rw_i+]	48 n:11ii	2
CMP	Rw_n, #$data_3$	48 n:0###	2
CMP	reg, #$data_{16}$	46 RR ## ##	4
CMP	reg, mem	42 RR MM MM	4

CMPB

Integer Compare

Syntax

CMPB op1, op2

Operation

(op1) ⇔ (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

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMPB	Rb_n, Rb_m	41 nm	2
CMPB	Rb_n, $[Rw_i]$	49 n:10ii	2
CMPB	Rb_n, $[Rw_i+]$	49 n:11ii	2
CMPB	Rb_n, #$data_3$	49 n:0###	2
CMPB	reg, #$data_{16}$	47 RR ## ##	4
CMPB	reg, mem	43 RR MM MM	4

CMPD1

Integer Compare and Decrement by 1

Syntax

CMPD1 op1, op2

Operation

(op1) ⇔ (op2)
(op1) ← (op1) - 1

Data Types

WORD

Description

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 decremented 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

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMPD1	Rw_n, $\#data_4$	A0 #n	2
CMPD1	Rw_n, $\#data_{16}$	A6 Fn ## ##	4
CMPD1	Rw_n, mem	A2 Fn MM MM	4

CMPD2

Integer Compare and Decrement by 2

Syntax

CMPD2 op1, op2

Operation

(op1) $\Leftrightarrow$ (op2)
(op1) $\leftarrow$ (op1) - 2

Data Types

WORD

Description

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 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

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMPD2	Rw_n, $\#data_4$	B0 #n	2
CMPD2	Rw_n, $\#data_{16}$	B6 Fn ## ##	4
CMPD2	Rw_n, mem	B2 Fn MM MM	4

CMPI1

Integer Compare and Increment by 1

Syntax

CMPI1 op1, op2

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 1

Data Types

WORD

Description

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

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMPI1	Rw_n, $\#data_4$	80 #n	2
CMPI1	Rw_n, $\#data_{16}$	86 Fn ## ##	4
CMPI1	Rw_n, mem	82 Fn MM MM	4

CMPI2

Integer Compare and Increment by 2

Syntax

CMPI2 op1, op2

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 2

Data Types

WORD

Description

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 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	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CMPI2	Rw_n, $\#data_4$	90 #n	2
CMPI2	Rw_n, $\#data_{16}$	96 Fn ## ##	4
CMPI2	Rw_n, mem	92 Fn MM MM	4

CPL

Integer One's Complement

Syntax

CPL op1

Operation

(op1) ← ¬(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	Z	V	C	N
*	*	0	0	*

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 Always cleared.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CPL	Rw_n	91 n0	2

CPLB

Integer One's Complement

Syntax

CPL op1

Operation

(op1) ← ¬(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	V	C	N
*	*	0	0	*

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 Always cleared.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CPLB	Rb_n	B1 n0	2

DISWDT

Disable Watchdog Timer

Syntax

DISWDT

Operation

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

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
DISWDT	A5 5A A5 A5	4

DIV

16-by-16 Signed Division

Syntax

DIV op1

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (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

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

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.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIV	Rw_n	4B nn	2

DIVL

32-by-16 Signed Division

Syntax

DIVL op1

Operation

(MDL) ← (MD) / (op1)
(MDH) ← (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

E	Z	V	C	N
0	*	S	0	*

E	Always cleared.
Z	Set if result equals zero. Cleared otherwise.
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.
C	Always cleared.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVL	Rw_n	6B nn	2

DIVLU

32-by-16 Unsigned Division

Syntax

DIVLU op1

Operation

(MDL) ← (MD) / (op1)
(MDH) ← (MD) mod (op1)

Data Types

WORD, DOUBLEWORD

Description

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).

Condition Flags	E	Z	V	C	N
	0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

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.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVLU	Rw_n	7B nn	2

DIVU

16-by-16 Unsigned Division

Syntax

DIVU op1

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Data Types

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

E	Z	V	C	N
0	*	S	0	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

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.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
DIVU	Rw_n	5B nn	2

EINIT

End of Initialization

Syntax

EINIT

Operation

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 $\overline{\text{RSTOUT}}$ 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 instruction is not accidentally executed, it is implemented as a protected instruction.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
EINIT	B5 4A B5 B5	4

EXTR

Begin EXTended Register Sequence

Syntax

EXTR op1

Operation

(count) ← (op1) [1 ≤ op1 ≤ 4]
Disable interrupts and Class A traps
SFR_range = Extended
DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) ← (count) - 1
END WHILE
(count) = 0
SFR_range = Standard
Enable 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 EXTR instruction must be used carefully (see Section 4.6 “Notes on ATOMIC and EXTended instructions”).

The EXTR instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTR	$\#data_2$	D1 :10##-0	2

EXTP

Begin EXTended Page Sequence

Syntax

EXTP op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page = (op1)
DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) ← (count) - 1
END WHILE
(count) = 0
Data_Page = (DPPx)
Enable 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 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 instruction sequence.

Note

The EXTP instruction must be used carefully (seeSection 4.6 “Notes on ATOMIC and EXTended instructions”).

The EXTP instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTP	Rwm, #$data_2$	DC :01##-m	2
EXTP	#pag, #$data_2$	D7 :01##-0 pp 0:00pp	4

EXTPR

Begin EXTended Page and Register Sequence

Syntax

EXTPR op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page = (op1) AND SFR_range = Extended
DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) ← (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. 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 instruction sequence.

Note

The EXTPR instruction must be used carefully (see Section 4.6 “Notes on ATOMIC and EXTended instructions”).
The EXTPR instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTPR	Rwm, #$data_2$	DC :11##-m	2
EXTPR	#pag, #$data_2$	D7 :11##-0 pp 0:00pp	4

EXTS

Begin EXTended Segment Sequence

Syntax

EXTS op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment = (op1)
DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) ← (count) - 1
END WHILE
(count) = 0
Data_Page = (DPPx)
Enable 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 sequence.

Note

The EXTS instruction must be used carefully (see Section 4.6 “Notes on ATOMIC and EXTended instructions”).

The EXTS instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTS	Rwm, $\#data_2$	DC :00##-m	2
EXTS	#seg, $\#data_2$	D7 :00##-0 ss 00	4

EXTSR

Begin EXTended Segment and Register Sequence

Syntax

EXTSR op1, op2

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment = (op1) AND SFR_range = Extended
DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) ← (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 instruction sequence.

Note

The EXTSR instruction must be used carefully (see Section 4.6 “Notes on ATOMIC and EXTended instructions”).
The EXTSR instruction is not available for ST10X166 devices.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
EXTSR	Rwm, #$data_2$	DC :10##-m	2
EXTSR	#seg, #$data_2$	D7 :10##-0 ss 00	4

IDLE

Enter Idle Mode

Syntax

IDLE

Operation

Enter Idle Mode

Description

This instruction causes the part to enter the idle mode. In this mode, 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

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
IDLE	87 78 87 87	4

JB

Relative Jump if Bit Set

Syntax

JB op1, op2

Operation

IF (op1) = 1 THEN
 (IP) ← (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 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 JB instruction. If the specified bit is clear, the instruction following the JB instruction is executed.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JB	$bitaddr_{Q.q}$, rel	8A QQ rr q0	4

JBC

Relative Jump if Bit Set and Clear Bit

Syntax

JBC op1, op2

Operation

IF (op1) = 1 THEN
 (op1) = 0
 (IP) ← (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

E	Z	V	C	N
-	$\overline{\mathbf{B}}$	-	-	B

E Not affected.

Z Contains logical negation of the previous state of the specified bit.

V Not affected.

C Not affected.

N Contains the previous state of the specified bit.

Addressing Modes

Mnemonic		Format	Bytes
JBC	$\text{bitaddr}_{Q.q}$, rel	AA QQ rr q0	4

JMPA

Absolute Conditional Jump

Syntax

JMPA op1, op2

Operation

IF (op1) = 1 THEN
 (IP) ← op2
w
 Next Instruction
END IF

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 instruction is executed normally.

Condition Codes

See condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPA	cc, caddr	EA c0 MM MM	4

JMPI

Indirect Conditional Jump

Syntax

JMPI op1, op2

Operation

IF (op1) = 1 THEN
(IP) $\leftarrow$ (op2)
ELSE
Next Instruction
END IF

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 instruction is executed normally.

Condition Codes

See condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPI	cc, [Rw_n]	9C cn	2

JMPR

Relative Conditional Jump

Syntax

JMPR op1, op2

Operation

IF (op1) = 1 THEN
(IP) ← (IP) + sign_extend (op2)
ELSE
Next Instruction
END IF

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.

Condition Codes

See condition code Table 4.17 on page 45.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPR	cc, rel	cD rr	2

JMPS

Absolute Inter-Segment Jump

Syntax

JMPS op1, op2

Operation

(CSP) ← op1
(IP) ← op2

Description

Branches unconditionally to the absolute address specified by op2 within the segment specified by op1.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JMPS	seg, caddr	FA ss MM MM	4

JNB

Relative Jump if Bit Clear

Syntax

JNB op1, op2

Operation

IF (op1) = 0 THEN
 (IP) ← (IP) + sign_extend (op2)
ELSE
 Next Instruction
END 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 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 JNB instruction. If the specified bit is set, the instruction following the JNB instruction is executed.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
JNB	bitaddr$_{Q.q}$, rel	9A QQ rr q0	4

JNBS

Relative Jump if Bit Clear and Set Bit

Syntax

JNBS op1, op2

Operation

IF (op1) = 0 THEN
 (op1) = 1
 (IP) ← (IP) + sign_extend (op2)
ELSE
 Next Instruction
END 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	Z	V	C	N
-	B	-	-	B

E Not affected.

Z Contains logical negation of the previous state of the specified bit.

V Not affected.

C Not affected.

N Contains the previous state of the specified bit.

Addressing Modes

Mnemonic		Format	Bytes
JNBS	$bitaddr_{Q.q}$, rel	BA QQ rr q0	4

MOV

Move Data

Syntax

MOV op1, op2

Operation

(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

E	Z	V	C	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 the value of the source operand op2 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
MOV	Rw_n, Rw_m	F0 nm	2
MOV	Rw_n, #$data_4$	E0 #n	2
MOV	reg, #$data_{16}$	E6 RR ## ##	4
MOV	Rw_n, [Rw_m]	A8 nm	2
MOV	Rw_n, [Rw_m+]	98 nm	2
MOV	[Rw_m], Rw_n	B8 nm	2
MOV	[-Rw_m], Rw_n	88 nm	2
MOV	[Rw_n], [Rw_m]	C8 nm	2
MOV	[Rw_n+], [Rw_m]	D8 nm	2
MOV	[Rw_n], [Rw_m+]	E8 nm	2
MOV	Rw_n, [Rw_m+#$data_{16}$]	D4 nm ## ##	4
MOV	[Rw_m+#$data_{16}$], Rw_n	C4 nm ## ##	4
MOV	[Rw_n], mem	84 0n MM MM	4
MOV	mem, [Rw_n]	94 0n MM MM	4
MOV	reg, mem	F2 RR MM MM	4
MOV	mem, reg	F6 RR MM MM	4

MOVB

Move Data

Syntax

MOVB op1, op2

Operation

(op1) ← (op2)

Data Types

BYTE

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

E	Z	V	C	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 the value of the source operand op2 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
MOVB	Rb_n, Rb_m	F1 nm	2
MOVB	Rb_n, $\#data_4$	E1 #n	2
MOVB	reg, $\#data_{16}$	E7 RR ## ##	4
MOVB	Rb_n, [Rw_m]	A9 nm	2
MOVB	Rb_n, [Rw_m+]	99 nm	2
MOVB	[Rw_m], Rb_n	B9 nm	2
MOVB	[-Rw_m], Rb_n	89 nm	2
MOVB	[Rw_n], [Rw_m]	C9 nm	2
MOVB	[Rw_n+], [Rw_m]	D9 nm	2
MOVB	[Rw_n], [Rw_m+]	E9 nm	2
MOVB	Rb_n, [Rw_m+$\#data_{16}$]	F4 nm ## ##	4
MOVB	[Rw_m+$\#data_{16}$], Rb_n	E4 nm ## ##	4
MOVB	[Rw_n], mem	A4 0n MM MM	4
MOVB	mem, [Rw_n]	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 op1, op2

Operation

(low byte op1) $\leftarrow$ (op2)
IF ($op2_7$) = 1 THEN
 (high byte op1) $\leftarrow$ FF_H
ELSE
 (high byte op1) $\leftarrow$ 00_H
END IF

Data Types

WORD, BYTE

Description

Moves and sign extends the contents of the source byte specified by op2 to the word 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
0	*	-	-	*

E Always cleared.

Z Set if the value of the source operand op2 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
MOVBS	Rw_n, Rb_m	D0 mn	2
MOVBS	reg, mem	D2 RR MM MM	4
MOVBS	mem, reg	D5 RR MM MM	4

MOVBZ

Move Byte Zero Extend

Syntax

MOVBZ op1, op2

Operation

(low byte op1) $\leftarrow$ (op2)
(high byte op1) $\leftarrow$ 00_H

Data Types

WORD, BYTE

Description

Moves and zero extends the contents of the source byte specified by op2 to the word 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
0	*	-	-	0

E Always cleared.

Z Set if the value of the source operand op2 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Always cleared.

Addressing Modes

Mnemonic		Format	Bytes
MOVBZ	Rw_n, Rb_m	C0 mn	2
MOVBZ	reg, mem	C2 RR MM MM	4
MOVBZ	mem, reg	C5 RR MM MM	4

MUL

Signed Multiplication

Syntax

MUL op1, op2

Operation

(MD) $\leftarrow$ (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

E	Z	V	C	N
0	*	S	0	0

E Always cleared.

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.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
MUL	Rw_n, Rw_m	0B nm	2

MULU

Unsigned Multiplication

Syntax

MULU op1, op2

Operation

(MD) ← (op1) * (op2)

Data Types

WORD

Description

Performs a 16-bit by 16-bit unsigned multiplication using the two words specified by operands op1 and op2 respectively. The unsigned 32-bit result is placed in the MD register.

Condition Flags

E	Z	V	C	N
0	*	S	0	0

E	Always cleared.
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.
C	Always cleared.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
MULU	Rw_n, Rw_m	1foB nm	2

NEG

Integer Two's Complement

Syntax

NEG op1

Operation

(op1) $\leftarrow$ 0 - (op1)

Data Types

WORD

Description

Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

Condition Flags

E	Z	V	C	N
*	*	*	S	*

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.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
NEG	Rw_n	81 n0	2

NEGB

Integer Two's Complement

Syntax

NEGB op1

Operation

(op1) $\leftarrow$ 0 - (op1)

Data Types

BYTE

Description

Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

Condition Flags

E	Z	V	C	N
*	*	*	S	*

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.

C — Set if a borrow is generated. Cleared otherwise.

N — Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
NEGB	Rb_n	A1 n0	2

NOP

No Operation

Syntax

NOP

Operation

No Operation

Description

This instruction causes a null operation to be performed. A null operation causes no change in the status of the flags.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
NOP	CC 00	2

OR

Logical OR

Syntax

OR op1, op2

Operation

(op1) $\leftarrow$ (op1) $\vee$ (op2)

Data Types

WORD

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	C	N
*	*	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.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
OR	Rw_n, Rw_m	70 nm	2
OR	Rw_n, [Rw_i]	78 n:10ii	2
OR	Rw_n, [Rw_i+]	78 n:11ii	2
OR	Rw_n, #$data_3$	78 n:0###	2
OR	reg, #$data_{16}$	76 RR ## ##	4
OR	reg, mem	72 RR MM MM	4
OR	mem, reg	74 RR MM MM	4

ORB

Logical OR

Syntax

ORB op1, op2

Operation

(op1) $\leftarrow$ (op1) $\vee$ (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	C	N
*	*	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.
V	Always cleared.
C	Always cleared.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ORB	Rb_n, Rb_m	71 nm	2
ORB	Rb_n, [Rw_i]	79 n:10ii	2
ORB	Rb_n, [Rw_i+]	79 n:11ii	2
ORB	Rb_n, #$data_3$	79 n:0###	2
ORB	reg, #$data_{16}$	77 RR ## ##	4
ORB	reg, mem	73 RR MM MM	4
ORB	mem, reg	75 RR MM MM	4

PCALL

Push Word and Call Subroutine Absolute

Syntax

PCALL op1, op2

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← op2

Data Types

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	V	C	N
*	*	-	-	*

E Set if the value of the pushed operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the pushed operand op1 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the pushed operand op1 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
CALL	reg, caddr	E2 RR MM MM	4

POP

Pop Word from System Stack

Syntax

POP op1

Operation

(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

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 popped word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the popped word equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the popped word is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
POP	reg	FC RR	2

PRIOR

Prioritize Register

Syntax

PRIOR op1, op2

Operation

(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)

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

E	Z	V	C	N
0	*	0	0	0

E Always cleared.

Z Set if the source operand op2 equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Always cleared.

Addressing Modes

Mnemonic		Format	Bytes
PRIOR	Rw_n, Rw_m	2B nm	2

PUSH

Push Word on System Stack

Syntax

PUSH op1

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)

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
*	*	-	-	*

E Set if the value of the pushed word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the pushed word equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the pushed word is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
PUSH	reg	EC RR	2

PWRDN

Enter Power Down Mode

Syntax

PWRDN

Operation

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.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
PWRDN	97 68 97 97	4

RET

Return from Subroutine

Syntax

RET

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2

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

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
RET	CB 00	2

RETI

Return from Interrupt Routine

Syntax

RETI

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
IF (SYSCON.SGTDIS=0) THEN
 (CSP) ← ((SP))
 (SP) ← (SP) + 2
END IF
(PSW) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from an interrupt routine. The PSW, IP, and CSP are popped off the system stack. Execution resumes at the instruction which had been interrupted. The previous system state is restored after the PSW has been popped. The CSP is only popped if segmentation is enabled. This is indicated by the SGTDIS bit in the SYSCON register.

Condition Flags

E	Z	V	C	N
S	S	S	S	S

E	Restored from the PSW popped from stack.
Z	Restored from the PSW popped from stack.
V	Restored from the PSW popped from stack.
C	Restored from the PSW popped from stack.
N	Restored from the PSW popped from stack.

Addressing Modes

Mnemonic	Format	Bytes
RETI	FB 88	2

RETP

Return from Subroutine and Pop Word

Syntax

RETP op1

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

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

E	Z	V	C	N
*	*	-	-	*

E Set if the value of the word popped into operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the word popped into operand op1 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the word popped into operand op1 is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
RETP	reg	EB RR	2

RETS

Return from Inter-Segment Subroutine

Syntax

RETS

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(CSP) ← ((SP))
(SP) ← (SP) + 2

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

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Mode

Mnemonic	Format	Bytes
RETS	DB 00	2

ROL

Rotate Left

Syntax

ROL op1, op2

Operation

(count) $\leftarrow$ (op2)
(C) $\leftarrow$ 0
DO WHILE (count) $\neq$ 0
(C) $\leftarrow$ ($op1_{15}$)
($op1_n$) $\leftarrow$ ($op1_{n-1}$) [n=1...15]
($op1_0$) $\leftarrow$ (C)
(count) $\leftarrow$ (count) - 1
END WHILE

Data Types

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

E	Z	V	C	N
0	*	0	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Always cleared.

C The carry flag is set according to the last MSB shifted out of op1. Cleared for a rotate count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ROL	Rw_n, Rw_m	0C nm	2
ROL	Rw_n, $\#data_4$	1C #n	2

ROR

Rotate Right

Syntax

ROR op1, op2

Operation

(count) $\leftarrow$ (op2)
(C) $\leftarrow$ 0
(V) $\leftarrow$ 0
DO WHILE (count) $\neq$ 0
 (V) $\leftarrow$ (V) $\vee$ (C)
 (C) $\leftarrow$ ($op1_0$)
 ($op1_n$) $\leftarrow$ ($op1_{n+1}$) [n=0...14]
 ($op1_{15}$) $\leftarrow$ (C)
 (count) $\leftarrow$ (count) - 1
END WHILE

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

E	Z	V	C	N
0	*	S	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if in any cycle of the rotate operation a ‘1’ is shifted out of the carry flag. Cleared for a rotate count of zero.

C The carry flag is set according to the last LSB shifted out of op1. Cleared for a rotate count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
ROR	Rw_n, Rw_m	2C nm	2
ROR	Rw_n, $\#data_4$	3C #n	2

SCXT

Switch Context

Syntax

SCXT op1, op2

Operation

(tmp1) ← (op1)
(tmp2) ← (op2)
(SP) ← (SP) - 2
((SP)) ← (tmp1)
(op1) ← (tmp2)

Description

Used to switch contexts for any register. Switching context is a push and load operation. The contents of the register specified by the first operand, op1, are pushed onto the stack. That register is then loaded with the value specified by the second operand, op2.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic		Format	Bytes
SCXT	reg, #$data_{16}$	C6 RR ## ##	4
SCXT	reg, mem	D6 RR MM MM	4

SHL

Shift Left

Syntax

SHL op1, op2

Operation

(count) ← (op2)
(C) ← 0
DO WHILE (count) ≠ 0
 (C) ← ($op1_{15}$)
 ($op1_n$) ← ($op1_{n-1}$) [n=1...15]
 ($op1_0$) ← 0
 (count) ← (count) - 1
END WHILE

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

E	Z	V	C	N
0	*	0	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Always cleared.

C The carry flag is set according to the last MSB shifted out of op1. Cleared for a shift count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SHL	Rw_n, Rw_m	4C nm	2
SHL	Rw_n, $\#data_4$	5C #n	2

SHR

Shift Right

Syntax

SHR op1, op2

Operation

(count) $\leftarrow$ (op2)
(C) $\leftarrow$ 0
(V) $\leftarrow$ 0
DO WHILE (count) $\neq$ 0
 (V) $\leftarrow$ (C) $\vee$ (V)
 (C) $\leftarrow$ ($op1_0$)
 ($op1_n$) $\leftarrow$ ($op1_{n+1}$) [n=0...14]
 ($op1_{15}$) $\leftarrow$ 0
 (count) $\leftarrow$ (count) - 1
END WHILE

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 significant 4 bits are used.

Condition Flags

E	Z	V	C	N
0	*	S	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V 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 shifted out of op1. Cleared for a shift count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SHR	Rw_n, Rw_m	6C nm	2
SHR	Rw_n, $\#data_4$	7C #n	2

SRST

Software Reset

Syntax

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 externally 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
0	0	0	0	0

E Always cleared.

Z Always cleared.

V Always cleared.

C Always cleared.

N Always cleared.

Addressing Modes

Mnemonic	Format	Bytes
SRST	B7 48 B7 B7	4

SRVWDT

Service Watchdog Timer

Syntax

SRVWDT

Operation

Service Watchdog Timer

Description

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.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.

Z Not affected.

V Not affected.

C Not affected.

N Not affected.

Addressing Modes

Mnemonic	Format	Bytes
SRVWDT	A7 58 A7 A7	4

SUB

Integer Subtraction

Syntax

SUB op1, op2

Operation

(op1) ← (op1) - (op2)

Data Types

WORD

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 from the destination operand specified by op1. The result is then stored in op1.

Condition Flags

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.
C	Set if a borrow is generated. Cleared otherwise.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SUB	Rw_n, Rw_m	20 nm	2
SUB	Rw_n, [Rw_i]	28 n:10ii	2
SUB	Rw_n, [Rw_i+]	28 n:11ii	2
SUB	Rw_n, #$data_3$	28 n:0###	2
SUB	reg, #$data_{16}$	26 RR ## ##	4
SUB	reg, mem	22 RR MM MM	4
SUB	mem, reg	24 RR MM MM	4

SUBB

Integer Subtraction

Syntax

SUBB op1, op2

Operation

(op1) ← (op1) - (op2)

Data Types

BYTE

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 from the destination operand specified by op1. The result is then stored in op1.

Condition Flags

Condition Flags	E	Z	V	C	N
	*	*	*	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.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SUBB	Rb_n, Rb_m	21 nm	2
SUBB	Rb_n, $[Rw_i]$	29 n:10ii	2
SUBB	Rb_n, $[Rw_i+]$	29 n:11ii	2
SUBB	Rb_n, #$data_3$	29 n:0###	2
SUBB	reg, #$data_{16}$	27 RR ## ##	4
SUBB	reg, mem	23 RR MM MM	4
SUBB	mem, reg	25 RR MM MM	4

SUBC

Integer Subtraction with Carry

Synta

SUBC op1, op2

Operation

(op1) ← (op1) - (op2) - (C)

Data Types

WORD

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.

Condition Flags

Condition Flags	E	Z	V	C	N
	*	S	*	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 and the previous Z flag was set. Cleared otherwise.

V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the specified data type. Cleared otherwise.

C Set if a borrow is generated. Cleared otherwise.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SUBC	Rw_n, Rw_m	30 nm	2
SUBC	Rw_n, [Rw_i]	38 n:10ii	2
SUBC	Rw_n, [Rw_i+]	38 n:11ii	2
SUBC	Rw_n, #$data_3$	38 n:0###	2
SUBC	reg, #$data_{16}$	36 RR ## ##	4
SUBC	reg, mem	32 RR MM MM	4
SUBC	mem, reg	34 RR MM MM	4

SUBCB

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.

Condition Flags

E	Z	V	C	N
*	*	*	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, i.e. the result cannot be represented in the specified data type. Cleared otherwise.
C	Set if a borrow is generated. Cleared otherwise.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
SUBCB	Rb_n, Rb_m	31 nm	2
SUBCB	Rb_n, [Rw_i]	39 n:10ii	2
SUBCB	Rb_n, [Rw_i+]	39 n:11ii	2
SUBCB	Rb_n, #$data_3$	39 n:0###	2
SUBCB	reg, #$data_{16}$	37 RR ## ##	4
SUBCB	reg, mem	33 RR MM MM	4
SUBCB	mem, reg	35 RR MM MM	4

TRAP

Software Trap

Syntax

TRAP op1

Operation

(SP) ← (SP) - 2
((SP)) ← (PSW)
IF (SYSCON.SGTDIS=0) THEN
 (SP) ← (SP) - 2
 ((SP)) ← (CSP)
 (CSP) ← 0
END IF
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← zero_extend (op1*4)

Description

Invokes a trap or interrupt routine based on the specified operand, op1. The invoked routine is determined by branching to the specified vector table entry point. This routine has no indication of whether it was called by software or hardware. System state is preserved identically to hardware interrupt entry except that the CPU priority level is not affected. The RETI, return from interrupt, instruction is used to resume execution after the trap or interrupt routine has completed. The CSP is pushed if segmentation is enabled. This is indicated by the SGTDIS bit in the SYSCON register.

Condition Flags

E	Z	V	C	N
-	-	-	-	-

E	Not affected.
Z	Not affected.
V	Not affected.
C	Not affected.
N	Not affected.

Addressing Modes

Mnemonic		Format	Bytes
TRAP	#trap7	9B t:ttt0	2

XOR

Logical Exclusive OR

Syntax

XOR op1, op2

Operation

(op1) $\leftarrow$ (op1) $\oplus$ (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

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 signal the end of a table.
Z	Set if result equals zero. Cleared otherwise.
V	Always cleared.
C	Always cleared.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
XOR	Rw_n, Rw_m	50 nm	2
XOR	Rw_n, [Rw_i]	58 n:10ii	2
XOR	Rw_n, [Rw_i+]	58 n:11ii	2
XOR	Rw_n, #$data_3$	58 n:0###	2
XOR	reg, #$data_{16}$	56 RR ## ##	4
XOR	reg, mem	52 RR MM MM	4
XOR	mem, reg	54 RR MM MM	4

XORB

Logical Exclusive OR

Syntax

XORB op1, op2

Operation

(op1) $\leftarrow$ (op1) $\oplus$ (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 signal the end of a table.

Z Set if result equals zero. Cleared otherwise.

V Always cleared.

C Always cleared.

N Set if the most significant bit of the result is set. Cleared otherwise.

Addressing Modes

Mnemonic		Format	Bytes
XORB	Rb_n, Rb_m	51 nm	2
XORB	Rb_n, [Rw_i]	59 n:10ii	2
XORB	Rb_n, [Rw_i+]	59 n:11ii	2
XORB	Rb_n, #$data_3$	59 n:0###	2
XORB	reg, #$data_{16}$	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_j). The GPR pointer allows access to the entire memory space, whereas IDX_i 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.

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

Symbol	Mnemonic	Address Pointer Operation
"[IDX_i⊗]" stands for	[IDX_i]	(IDX_i) ← (IDX_i) (no-op)
	[IDX_i+]	(IDX_i) ← (IDX_i) +2 (i=0,1)
	[IDX_i -]	(IDX_i) ← (IDX_i) -2 (i=0,1)
	[IDX_i + QX_j]	(IDX_i) ← (IDX_i) + (QX_j) (i, j =0,1)
	[IDX_i - QX_j]	(IDX_i) ← (IDX_i) - (QX_j) (i, j =0,1)
"[Rw_n⊗]" stands for	[Rwn]	(Rwn) ← (Rwn) (no-op)
	[Rwn+]	(Rwn) ← (Rwn) +2 (n=0-15)
	[Rwn-]	(Rwn) ← (Rwn) -2 (k=0-15)
	[Rwn + QR_j]	(Rwn) ← (Rwn) + (QR_j) (n=0-15;j =0,1)
	[Rwn - QR_j]	(Rwn) ← (Rwn) - (QR_j) (n=0-15; j =0,1)

In this document the symbols "[Rw_n⊗]" and "[IDX_i⊗]" are used to refer to these addressing modes.

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.

Table 6.2 MAC register addresses for CoReg

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

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

Table 8.1 MAC instruction mnemonics by addressing mode and repeatability

Mnemonic	Addressing Modes	Rep
CoMUL CoMULu CoMULus CoMULsu CoMUL- CoMULu- CoMULus- CoMULsu- CoMUL + rnd CoMULu + rnd CoMULus + rnd CoMULsu + rnd	Rw_n, Rw_m $[IDX_i\otimes]$, $[Rw_m\otimes]$ Rw_n, $[Rw_m\otimes]$	No No No
CoMAC CoMACu CoMACus CoMACsu CoMAC- CoMACu- CoMACus- CoMACsu- CoMAC + rnd CoMACu + rnd CoMACus + rnd CoMACsu + rnd CoMACR CoMACRu CoMACRus CoMACRsu CoMACR + rnd CoMACRu + rnd CoMACRus + rnd CoMACRsu + rnd	Rw_n, Rw_m $[IDX_i\otimes]$, $[Rw_m\otimes]$ Rw_n, $[Rw_m\otimes]$	No Yes Yes
CoNOP	$[Rw_m\otimes]$ $[IDX_i\otimes]$ $[IDX_i\otimes]$, $[Rw_m\otimes]$	Yes Yes Yes
CoNEG CoNEG + rnd CoRND	-	No
CoSTORE	Rw_n , CoReg $[Rw_n\otimes]$, Coreg	No Yes
CoMOV	$[IDX_i\otimes]$, $[Rw_m\otimes]$	Yes
CoMACM CoMACMu CoMACMus CoMACMsu CoMACM- CoMACMu- CoMACMus- CoMACMsu- CoMACM + rnd CoMACMu + rnd CoMACMus + rnd CoMACMsu + rnd CoMACMR CoMACMRu CoMACMRus CoMACMRsu CoMACMR + rnd CoMACMRu + rnd CoMACMRus + rnd CoMACMRsu + rnd	$[IDX_i\otimes]$, $[Rw_m\otimes]$	Yes
CoADD CoADD2 CoSUB CoSUB2 CoSUBR CoSUB2R CoMAX CoMIN	Rw_n, Rw_m $[IDX_i\otimes]$, $[Rw_m\otimes]$ Rw_n, $[Rw_m\otimes]$	No Yes Yes
CoLOAD CoLOAD- CoLOAD2 CoLOAD2- CoCMP	Rw_n, Rw_m $[IDX_i\otimes]$, $[Rw_m\otimes]$ Rw_n, $[Rw_m\otimes]$	No No No
CoSHL CoSHR CoASHR CoASHR + rnd	Rw_m #data4 $[Rw_m\otimes]$	Yes No Yes
CoABS	- Rw_n, Rw_m $[IDX_i\otimes]$, $[Rw_m\otimes]$ Rw_n, $[Rw_m\otimes]$	No No No No

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.

Table 8.2 MAC instruction function code (hexa)

Mnemonic	Function Code		Mnemonic	Function Code
CoMUL	C0		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		CoMAX	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	wwww:w000		CoASHR other	AA
CoMOV	00		CoASHR + rnd #d4	B2
			CoASHR + rnd other	BA

8.1 MAC instruction conventions

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

8.1.1 Operands

opX	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX_n)	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_H

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) $>>_a$	opX Arithmetically SHIFTED Right
opX\opY	opX (MSW) and opY (LSW) CONCATENATED to a 32-bit operand.

8.1.3 Data addressing modes

"Rw_n", or "Rw_m" :	General Purpose Registers (GPRs) where "n" and "m" are any values contained between 0 and 15 included.
[...] :	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').

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:

X	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.

Table 8.3 IDX Addressing Mode Encoding and GPR offset Encoding

Addressing Mode	4-bit Encoding
IDX0	1_h
IDX0 +	2_h
IDX0 -	3_h
IDX0 + QX0	4_h
IDX0 - QX0	5_h
IDX0 + QX1	6_h
IDX0 - QX1	7_h
IDX1	9_h
IDX1 +	A_h
IDX1 -	B_h
IDX1 + QX0	C_h
IDX1 - QX0	D_h
IDX1 + QX1	E_h
IDX1 - QX1	F_h

GPR Offset	3-bit Encoding
no-op	1_h
+	2_h
-	3_h
+ QR0	4_h
- QR0	5_h
+ QR1	6_h
- QR1	7_h

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 ($MRW_{12\text{-}0}$) + 1 times, therefore the maximum number of times an instruction can be repeated is 8 192 (2^{13}) 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(-) Signed Multiply Optional Round

Group
Multiply/Multiply-Accumulate Instructions

Syntax
CoMUL op1, op2

Operation
IF (MP = 1) THEN
$(ACC) \leftarrow ((op1) * (op2)) << 2$
ELSE
$(ACC) \leftarrow (op1) * (op2)$
END IF

Syntax
CoMUL- op1, op2

Operation
IF (MP = 1) THEN
$(ACC) \leftarrow -\ (\ ((op1) * (op2)) << 2\ )$
ELSE
$(ACC) \leftarrow -\ (\ (op1) * (op2)\ )$
END IF

Syntax
CoMUL op1, op2, rnd

Operation
IF (MP = 1) THEN
$(ACC) \leftarrow ((op1) * (op2)) << 2 + 0000\ 8000_H$
ELSE
$(ACC) \leftarrow (op1) * (op2) + 0000\ 8000_H$
END IF
$(MAL) \leftarrow 0$

Data Types
DOUBLE WORD

Result
32-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 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-repeatable instruction allows up to two parallel memory reads

MAC Condition Flags

N	Z	C	SV	E	SL
*	*	0	-	*	*

N Set if the most significant bit of the result is set. Cleared otherwise.

Z Set if the result equals zero. Cleared otherwise.

C Always cleared.

SV	Not affected.
E	Always cleared when MP is cleared, otherwise, only set in case of 8000_H by 8000_H multiplication.
SL	Not affected when MP or MS are cleared, otherwise, only set in case of 8000_H by 8000_H multiplication.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoMUL	Rw_n, Rw_m	No	A3 nm C0 00	4
CoMUL-	Rw_n, Rw_m	No	A3 nm C8 00	4
CoMUL	Rw_n, Rw_m, rnd	No	A3 nm C1 00	4
CoMUL	[IDX_i⊗], [Rw_m⊗]	No	93 Xm C0 0:0qqq	4
CoMUL-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm C8 0:0qqq	4
CoMUL	[IDX_i⊗], [Rw_m⊗], rnd	No	93 Xm C1 0:0qqq	4
CoMUL	Rw_n, [Rw_m⊗]	No	83 nm C0 0:0qqq	4
CoMUL-	Rw_n, [Rw_m⊗]	No	83 nm C8 0:0qqq	4
CoMUL	Rw_n, [Rw_m⊗], 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
                                    ; (IDX0) ← (IDX0) + (QX1)
CoMUL-  [IDX1 -], [R15+QR0]         ; (ACC) ← -((IDX1))*((R15))
                                    ; (R15) ← (R15) + (QR0)
                                    ; (IDX1) ← (IDX1) - 2
CoMUL   [IDX1+QX0], [R9 - QR1], rnd ; (ACC) ← ((IDX1))*((R9)) + rnd
                                    ; (R9) ← (R9) - (QR1)
                                    ; (IDX1) ← (IDX1) + (QX0).
```

Multiplication Examples

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Z	C	SV	E	SL
MP=0, MS=x	8000_H	8000_H	0	00_H	4000_H	0000_H	0	0	0	-	0	-
MP=1, MS=0			0	00_H	8000_H	0000_H	0	0	0	-	1	-
MP=1, MS=1			0	00_H	$7FFF_H$	$FFFF_H$	0	0	0	-	0	1
MP=0, MS=x	$7FFF_H$	$7FFF_H$	0	00_H	$3FFF_H$	0001_H	0	0	0	-	0	-
MP=1, MS=x			0	00_H	$7FFE_H$	0002_H	0	0	0	-	0	-
MP=1, MS=x			1	00_H	$7FFE_H$	0000_H	0	0	0	-	0	-
MP=0, MS=x	4001_H	$F456_H$	0	FF_H	$FD15_H$	7456_H	1	0	0	-	0	-
MP=1, MS=x			0	FF_H	$FA2A_H$	$E8AC_H$	1	0	0	-	0	-
MP=0, MS=x			1	FF_H	$FD15_H$	0000_H	1	0	0	-	0	-
MP=1, MS=x			1	FF_H	$FA2B_H$	0000_H	1	0	0	-	0	-

CoMULu(-) Unsigned Multiply Optional Round

Group
Multiply/Multiply-Accumulate Instructions

Syntax
CoMULu op1, op2

Operation
(ACC) $\leftarrow$ (op1) $*$ (op2)

Syntax
CoMULu- op1, op2

Operation
(ACC) $\leftarrow$ - ((op1) $*$ (op2))

Syntax
CoMULu op1, op2, rnd

Operation
(ACC) $\leftarrow$ (op1) $*$ (op2) + 0000 8000_H
(MAL) $\leftarrow$ 0

Data Types
DOUBLE WORD

Result
32-bit unsigned value

Description
Multiply the two unsigned 16-bit source operands “op1” and “op2”. The unsigned 32-bit product is first zero-extended, and 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 of 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

N	Z	C	SV	E	SL
*	*	0	-	0	-

N	Set if the most significant bit of the result is set. Cleared otherwise.
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
CoMULu	Rw_n, Rw_m	No	A3 nm 00 00	4
CoMULu-	Rw_n, Rw_m	No	A3 nm 08 00	4
CoMULu	Rw_n, Rw_m, rnd	No	A3 nm 01 00	4
CoMULu	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 00 0:0qqq	4
CoMULu-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 08 0:0qqq	4
CoMULu	[IDX_i⊗], [Rw_m⊗], rnd	No	93 Xm 01 0:0qqq	4
CoMULu	Rw_n, [Rw_m⊗]	No	83 nm 00 0:0qqq	4
CoMULu-	Rw_n, [Rw_m⊗]	No	83 nm 08 0:0qqq	4
CoMULu	Rw_n, [Rw_m⊗], 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) ← (R0)*(R1) + rnd
CoMULu-   R2, [R6+]                 ; (ACC) ← -(R2)*((R6))
                                    ; (R6) ← (R6) + 2
CoMULu    [IDX0], [R11+]            ; (ACC) ← ((IDX0))*((R11))
                                    ; (R11) ← (R11) + 2
CoMULu-   [IDX1 -], [R15+QR0]       ; (ACC) ← -((IDX1))*((R15))
                                    ; (R15) ← (R15) + (QR0)
                                    ; (IDX1) ← (IDX1) - 2
CoMULu    [IDX0+QX0], [R9 -], rnd   ; (ACC) ← ((IDX0))*((R15] + rnd
                                    ; (R9) ← (R9) - 2
                                    ; (IDX0) ← (IDX0) + (QX0).
```

Multiplication Examples

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Z	C	SV	E	SL
MP=x, MS=x	8000_H	8000_H	x	00_H	4000_H	0000_H	0	0	0	-	0	-
MP=x, MS=x	$7FFF_H$	$7FFF_H$	0	00_H	$3FFF_H$	0001_H	0	0	0	-	0	-
			1	00_H	$3FFF_H$	0000_H	0	0	0	-	0	-
MP=x, MS=x	8001_H	$F456_H$	0	00_H	$7A2B_H$	$F456_H$	0	0	0	-	0	-
			1	00_H	$7A2C_H$	0000_H	0	0	0	-	0	-
MP=x, MS=0	$FFFF_H$	$FFFF_H$	0	00_H	$FFFE_H$	0001_H	0	0	0	-	0	-
MP=x, MS=1			0	00_H	$7FFF_H$	$FFFF_H$	0	0	0	-	0	-

CoMULsu(-) Mixed Multiply Optional Round

Group Multiply/Multiply-Accumulate Instructions

Syntax CoMULsu op1, op2

Operation (ACC) $\leftarrow$ (op1) $*$ (op2)

Syntax CoMULsu- op1, op2

Operation (ACC) $\leftarrow$ - ((op1) $*$ (op2))

Syntax CoMULsu 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 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 parallel memory reads.

MAC Condition Flags

N	Z	C	SV	E	SL
*	*	0	-	0	-

N Set if the most significant bit of the result is set. Cleared otherwise.

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_n, Rw_m	No	A3 nm 80 00	4
CoMULsu-	Rw_n, Rw_m	No	A3 nm 88 00	4
CoMULsu	Rw_n, Rw_m, rnd	No	A3 nm 81 00	4
CoMULsu	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 80 0:0qqq	4
CoMULsu-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 88 0:0qqq	4
CoMULsu	[IDX_i⊗], [Rw_m⊗], rnd	No	93 Xm 81 0:0qqq	4
CoMULsu	Rw_n, [Rw_m⊗]	No	83 nm 80 0:0qqq	4
CoMULsu-	Rw_n, [Rw_m⊗]	No	83 nm 88 0:0qqq	4
CoMULsu	Rw_n, [Rw_m⊗], 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) ← ((IDX0))*((R11))
                                       ; (R11) ← (R11) + 2
CoMULsu-  [IDX1 -], [R15]              ; (ACC) ← -((IDX1))*((R15))
                                       ; (IDX1) ← (IDX1) - 2
CoMULsu   [IDX0+QX0], [R9 - QR1], rnd  ; (ACC) ← ((IDX0))*((R9)) + rnd
                                       ; (R9) ← (R9) - (QR1)
                                       ; (IDX0) ← (IDX0) + (QX0).
```

Multiplication Examples

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Z	C	SV	E	SL
MP=x, MS=x	8000_H	8000_H	x	FF_H	$C000_H$	0000_H	1	0	0	-	0	-
MP=x, MS=x	$7FFF_H$	$7FFF_H$	0	00_H	$3FFF_H$	0001_H	0	0	0	-	0	-
			1	00_H	$3FFF_H$	0000_H	0	0	0	-	0	-
MP=x, MS=x	8001_H	$F456_H$	0	FF_H	$85D5_H$	$F456_H$	1	0	0	-	0	-
			1	FF_H	$85D6_H$	0000_H	1	0	0	-	0	-

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) $\leftarrow$ - ((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 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

N	Z	C	SV	E	SL
*	*	0	-	0	-

N Set if the most significant bit of the result is set. Cleared otherwise.

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_n, Rw_m	No	A3 nm 40 00	4
CoMULus-	Rw_n, Rw_m	No	A3 nm 48 00	4
CoMULus	Rw_n, Rw_m, rnd	No	A3 nm 41 00	4
CoMULus	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 40 0:0qqq	4
CoMULus-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 48 0:0qqq	4
CoMULus	[IDX_i⊗], [Rw_m⊗], rnd	No	93 Xm 41 0:0qqq	4
CoMULus	Rw_n, [Rw_m⊗]	No	83 nm 40 0:0qqq	4
CoMULus-	Rwn, [Rw_m⊗]	No	83 nm 48 0:0qqq	4
CoMULus	Rw_n, [Rw_m⊗], rnd	No	83 nm 41 0:0qqq	4

Examples

```
CoMULus    R0, R1, rnd                   ; (ACC) ← (R0)*(R1) + rnd
CoMULus-   R2, [R6+]                     ; (ACC) ← -(R2)*((R6))
                                         ; (R6) ← (R6) + 2
CoMULus    [IDX1+QX0], [R11+QR0]         ; (ACC) ← ((IDX1))*((R11))
                                         ; (R11) ← (R11) + (QR0)
                                         ; (IDX1) ← (IDX1) + (QX0)
CoMULus-   [IDX0], [R15]                 ; (ACC) ← -((IDX0))*((R15))
CoMULus    [IDX0+QX0], [R9 - QR1], rnd   ; (ACC) ← ((IDX0))*((R15] + rnd
                                         ; (R9) ← (R9) - (QR1)
                                         ; (IDX0) ← (IDX0) + (QX0).
```

Multiplication Examples

Cases	op 1	op 2	rnd	MAE	MAH	MAL	N	Z	C	SV	E	SL
MP=x, MS=x	8000_H	8000_H	x	FF_H	$C000_H$	0000_H	1	0	0	-	0	-
MP=x, MS=x	$7FFF_H$	$7FFF_H$	0	00_H	$3FFF_H$	0001_H	0	0	0	-	0	-
			1	00_H	$3FFF_H$	0000_H	0	0	0	-	0	-
MP=x, MS=x	8001_H	$F456_H$	0	FF_H	$FA2A_H$	0001_H	1	0	0	-	0	-
			1	FF_H	$FA2B_H$	0000_H	1	0	0	-	0	-

CoMAC(R/-) Multiply-Accumulate Optional Round

Group

Multiply/Multiply-Accumulate Instructions

Syntax

CoMAC op1, op2

Operation

IF (MP = 1) THEN
(tmp) ← ((op1) ∗ (op2)) << 2
(ACC) ← (ACC) + (tmp)
ELSE
(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) + (tmp)
END IF

Syntax

CoMAC op1, op2, rnd

Operation

IF (MP = 1) THEN
(tmp) ← ((op1) ∗ (op2)) << 2
(ACC) ← (ACC) + (tmp) + $0000\ 8000_H$
ELSE
(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) + (tmp) + $0000\ 8000_H$
END IF
(MAL) ← 0

Syntax

CoMAC- op1, op2

Operation

IF (MP = 1) THEN
(tmp) ← ((op1) ∗ (op2)) << 2
(ACC) ← (ACC) - (tmp)
ELSE
(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) - (tmp)
END IF

Syntax

CoMACR op1, op2

Operation

IF (MP = 1) THEN
(tmp) ← ((op1) ∗ (op2)) << 2
(ACC) ← (tmp) - (ACC)
ELSE
(tmp) ← (op1) ∗ (op2)
(ACC) ← (tmp) - (ACC)
END IF

Syntax

CoMACR op1, op2, rnd

Operation

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

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 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

N	Z	C	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.
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
CoMAC	Rw_n, Rw_m	No	A3 nm D0 00	4
CoMAC-	Rw_n, Rw_m	No	A3 nm E0 00	4
CoMAC	Rw_n, Rw_m, rnd	No	A3 nm D1 00	4
CoMACR	Rw_n, Rw_m	No	A3 nm F0 00	4
CoMACR	Rw_n, Rw_m, rnd	No	A3 nm F1 00	4
CoMAC	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm D0 rrrr:rqqq	4
CoMAC-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm E0 rrrr:rqqq	4
CoMAC	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm D1 rrrr:rqqq	4
CoMACR	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm F0 rrrr:rqqq	4
CoMACR	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm F1 rrrr:rqqq	4
CoMAC	Rw_n, [Rw_m⊗]	Yes	83 nm D0 rrrr:rqqq	4
CoMAC-	Rw_n, [Rw_m⊗]	Yes	83 nm E0 rrrr:rqqq	4
CoMAC	Rw_n, [Rw_m⊗], rnd	Yes	83 nm D1rrrr:rqqq	4
CoMACR	Rw_n, [Rw_m⊗]	Yes	83 nm F0 rrrr:rqqq	4
CoMACR	Rw_n, [Rw_m⊗], rnd	Yes	83 nm F1 rrrr:rqqq	4

Examples

```
CoMAC     R3, R4, rnd                        ; (ACC) ← (ACC)   + (R3)*(R4) + rnd
CoMAC-    R2, [R6+]                          ; (ACC) ← (ACC) - (R2)*((R6))
                                             ; (R6) ← (R6) + 2
CoMAC     [IDX0+QX0], [R11+QR0]              ; (ACC) ← (ACC) + ((IDX0))*((R11))
                                             ; (R11) ← (R11) + (QR0)
                                             ; (IDX0) ← (IDX0) + (QX0)
Repeat 3 times CoMAC [IDX1 - QX1], [R9+QR1]
                                             ; (ACC) ← (ACC) + ((IDX1))*((R9))
                                             ; (R9) ← (R9) + (QR1)
                                             ; (IDX1) ← (IDX1) - (QX1)
Repeat MRW times CoMAC - R3, [R7 - QR0]      ; (ACC) ← (ACC) - (R3)*((R7))
                                             ; (R7) ← (R7) - (QR0)
CoMACR    [IDX1], [R4+], rnd                 ; ACC) ← ((IDX1))*((R4)) - (Acc) + rnd
                                             ; (R4) ← (R4) + 2
```

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

Group

Multiply/Multiply-Accumulate Instructions

Syntax

CoMACu op1, op2

Operation

(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) + (tmp)

Syntax

CoMACu op1, op2, rnd

Operation

(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) + (tmp) + 0000 8000_H
(MAL) ← 0

Syntax

CoMACu- op1, op2

Operation

(tmp) ← (op1) ∗ (op2)
(ACC) ← (ACC) - (tmp)

Syntax

CoMACRu op1, op2

Operation

(tmp) ← (op1) ∗ (op2)
(ACC) ← (tmp) - (ACC)

Syntax

CoMACRu op1, op2, rnd

Operation

(tmp) ← (op1) ∗ (op2)
(ACC) ← (tmp) - (ACC) + 0000 8000_H
(MAL) ← 0

Data Types

DOUBLE WORD

Result

40-bit unsigned value

Description

Multiplies the two unsigned 16-bit source operands “op1” and “op2”. The obtained unsigned 32-bit product is first zero-extended and 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. 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

N	Z	C	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.
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
CoMACu	Rw_n, Rw_m	No	A3 nm 10 00	4
CoMACu-	Rw_n, Rw_m	No	A3 nm 20 00	4
CoMACu	Rw_n, Rw_m, rnd	No	A3 nm 11 00	4
CoMACRu	Rw_n, Rw_m	No	A3 nm 30 00	4
CoMACRu	Rw_n, Rw_m, rnd	No	A3 nm 31 00	4
CoMACu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 10 rrrr:rqqq	4
CoMACu-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 20 rrrr:rqqq	4
CoMACu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 11 rrrr:rqqq	4
CoMACRu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 30 rrrr:rqqq	4
CoMACRu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 31 rrrr:rqqq	4
CoMACu	Rw_n, [Rw_m⊗]	Yes	83 nm 10 rrrr:rqqq	4
CoMACu-	Rw_n, [Rw_m⊗]	Yes	83 nm 20 rrrr:rqqq	4
CoMACu	Rw_n, [Rw_m⊗], rnd	Yes	83 nm 11 rrrr:rqqq	4
CoMACRu	Rw_n, [Rw_m⊗]	Yes	83 nm 30 rrrr:rqqq	4
CoMACRu	Rw_n, [Rw_m⊗], rnd	Yes	83 nm 31 rrrr:rqqq	4

Examples

```
CoMACu      R5, R8, rnd                           ; (ACC) ← (ACC) + (R5)*(R8) + rnd
CoMACu-     R2, [R7]                              ; (ACC) ← (ACC) - (R2)*((R7))
CoMACu      [IDX0 - QX0], [R11 - QR0]             ; (ACC) ← (ACC) + ((IDX0))*((R11))
                                                  ; (R11) ← (R11) - (QR0)
                                                  ; (IDX0) ← (IDX0) - (QX0)
Repeat 3 times CoMACu [IDX1+], [R9 -]             ; (ACC) ← (ACC) + ((IDX1))*((R9))
                                                  ; (R9) ← (R9) - 2
                                                  ; (IDX1) ← (IDX1) + 2
Repeat MRW times CoMACu- R3, [R7 - QR0]           ; (ACC) ← (ACC) - (R3)*((R7))
                                                  ; (R7) ← (R7) - (QR0)
CoMACRu     [IDX1 - QX0], [R4], rnd               ; (ACC) ← ((IDX1))*((R4))-(ACC)+ rnd
                                                  ; (IDX1) ← (IDX1) - (QX0)
```

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

Group Multiply/Multiply-Accumulate Instructions

Syntax CoMACsu op1, op2

Operation
(tmp) $\leftarrow$ (op1) $*$ (op2)
(ACC) $\leftarrow$ (ACC) + (tmp)

Syntax CoMACsu op1, op2, rnd

Operation
(tmp) $\leftarrow$ (op1) $*$ (op2)
(ACC) $\leftarrow$ (ACC) + (tmp) + $0000\ 8000_H$
(MAL) $\leftarrow$ 0

Syntax CoMACsu- op1, op2

Operation
(tmp) $\leftarrow$ (op1) $*$ (op2)
(ACC) $\leftarrow$ (ACC) - (tmp)

Syntax CoMACRsu op1, op2

Operation
(tmp) $\leftarrow$ (op1) $*$ (op2)
(ACC) $\leftarrow$ (tmp) - (ACC)

Syntax CoMACRsu 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 signed and unsigned 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

N	Z	C	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.
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
CoMACsu	Rw_n, Rw_m	No	A3 nm 90 00	4
CoMACsu-	Rw_n, Rw_m	No	A3 nm A0 00	4
CoMACsu	Rw_n, Rw_m, rnd	No	A3 nm 91 00	4
CoMACRsu	Rw_n, Rw_m	No	A3 nm B0 00	4
CoMACRsu	Rw_n, Rw_m, rnd	No	A3 nm B1 00	4
CoMACsu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 90 rrrr:rqqq	4
CoMACsu-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm A0 rrrr:rqqq	4
CoMACsu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 91 rrrr:rqqq	4
CoMACRsu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm B0 rrrr:rqqq	4
CoMACRsu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm B1 rrrr:rqqq	4
CoMACsu	Rw_n, [Rw_m⊗]	Yes	83 nm 90 rrrr:rqqq	4
CoMACsu-	Rw_n, [Rw_m⊗]	Yes	83 nm A0 rrrr:rqqq	4
CoMACsu	Rw_n, [Rw_m⊗], rnd	Yes	83 nm 91 rrrr:rqqq	4
CoMACRsu	Rw_n, [Rw_m⊗]	Yes	83 nm B0 rrrr:rqqq	4
CoMACRsu	Rw_n, [Rw_m⊗], rnd	Yes	83 nm B1 rrrr:rqqq	4

Examples

```
CoMACsu     R5, R8, rnd                     ; (ACC) ← (ACC) + (R5)*(R8) + rnd
CoMACsu-    R2, [R7]                        ; (ACC) ← (ACC) - (R2)*((R7))
CoMACsu     [IDX0 - QX0], [R11 - QR0]       ; (ACC) ← (ACC) + ((IDX0))*((R11))
                                            ; (R11) ← (R11) - (QR0)
                                            ; (IDX0) ← (IDX0) - (QX0)
Repeat 3 times CoMACsu [IDX1+], [R9 -]      ; (ACC) ← (ACC) + ((IDX1))*((R9))
                                            ; (R9) ← (R9) - 2
                                            ; (IDX1) ← (IDX1) + 2
Repeat MRW times CoMACsu -R3, [R7 - QR0]
                                            ; (ACC) ← (ACC) - (R3)*((R7))
                                            ; (R7) ← (R7) - (QR0)
CoMACsur    [IDX1 - QX0], [R4], rnd         ; (ACC) ← ((IDX1))*((R4)) - (ACC)
                                            ; (IDX1) ← (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
$(tmp) \leftarrow (op1) * (op2)$
$(ACC) \leftarrow (ACC) + (tmp) + 0000\ 8000_H$
$(MAL) \leftarrow 0$

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

N	Z	C	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.

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
CoMACus	Rw_n, Rw_m	No	A3 nm 50 00	4
CoMACus-	Rw_n, Rw_m	No	A3 nm 60 00	4
CoMACus	Rw_n, Rw_m, rnd	No	A3 nm 51 00	4
CoMACRus	Rw_n, Rw_m	No	A3 nm 70 00	4
CoMACRus	Rw_n, Rw_m, rnd	No	A3 nm 71 00	4
CoMACus	[$IDX_i\otimes$], [$Rw_m\otimes$]	Yes	93 Xm 50 rrrr:rqqq	4
CoMACus-	[$IDX_i\otimes$], [$Rw_m\otimes$]	Yes	93 Xm 60 rrrr:rqqq	4
CoMACus	[$IDX_i\otimes$], [$Rw_m\otimes$], rnd	Yes	93 Xm 51 rrrr:rqqq	4
CoMACRus	[$IDX_i\otimes$], [$Rw_m\otimes$]	Yes	93 Xm 70 rrrr:rqqq	4
CoMACRus	[$IDX_i\otimes$], [$Rw_m\otimes$], rnd	Yes	93 Xm 71 rrrr:rqqq	4
CoMACus	Rw_n, [$Rw_m\otimes$]	Yes	83 nm 50 rrrr:rqqq	4
CoMACus-	Rw_n, [$Rw_m\otimes$]	Yes	83 nm 60 rrrr:rqqq	4
CoMACus	Rw_n, [$Rw_m\otimes$], rnd	Yes	83 nm 61 rrrr:rqqq	4
CoMACRus	Rw_n, [$Rw_m\otimes$]	Yes	83 nm 70 rrrr:rqqq	4
CoMACRus	Rw_n, [$Rw_m\otimes$], rnd	Yes	83 nm 71 rrrr:rqqq	4

Examples

```
CoMACus    R5, R8, rnd                      ; (ACC) ← (ACC) + (R5)*(R8) + rnd
CoMACus-   R2, [R7]                         ; (ACC) ← (ACC) - (R2)*((R7))
CoMACus    [IDX0 - QX0], [R11 - QR0]        ; (ACC) ← (ACC) + ((IDX0))*((R11))
                                            ; (R11) ← (R11) - (QR0)
                                            ; (IDX0) ← (IDX0) - (QX0)
Repeat 3 times CoMACus[IDX1+], [R9 -]       ; (ACC) ← (ACC) + ((IDX1))*((R9))
                                            ; (R9) ← (R9) - 2
                                            ; (IDX1) ← (IDX1) + 2
Repeat MRW times CoMACus-R3, [R7 - QR0]; (ACC) ← (ACC) - (R3)*((R7))
                                            ; (R7) ← (R7) - (QR0)
CoMACRus   [IDX1 - QX0], [R4], rnd          ;(ACC) ← ((IDX1))*((R4))-(ACC)+rnd
                                            ; (IDX1) ← (IDX1) - (QX0)
```

CoMACM(R/-)

Multiply-Accumulate
Parallel Data Move and Optional Round

Group
Multiply/Multiply-Accumulate Instructions

Syntax
CoMACM [IDX_i⊗], [Rw_m⊗]

Operation
IF (MP = 1) THEN
(tmp) ← ((IDX_i))*((Rw_m)) << 2
(ACC) ← (ACC) + (tmp)
ELSE
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp)
END IF ((IDX_i(-⊗))) ← ((IDX_i))

Syntax
CoMACM [IDX_i⊗], [Rw_m⊗], rnd

Operation
IF (MP = 1) THEN
(tmp) ← ((IDX_i))*((Rw_m)) << 2
(ACC) ← (ACC) + (tmp) + $0000\ 8000_H$
ELSE
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp) + $0000\ 8000_H$
END IF
(MAL) ← 0 ((IDX_i(-⊗))) ← ((IDX_i))

Syntax
CoMACM- [IDX_i⊗], [Rw_m⊗]

Operation
IF (MP = 1) THEN
(tmp) ← ((IDX_i))*((Rw_m)) << 2
(ACC) ← (ACC) + (tmp)
ELSE
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) - (tmp)
END IF ((IDX_i(-⊗))) ← ((IDX_i))

Syntax
CoMACMR [IDX_i⊗], [Rw_m⊗]

Operation
IF (MP = 1) THEN
(tmp) ← ((IDX_i))*((Rw_m)) << 2
(ACC) ← (-ACC) + (tmp)
ELSE
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (-ACC) + (tmp)
END IF ((IDX_i(-⊗))) ← ((IDX_i))

Syntax
CoMACMR [IDX_i⊗], [Rw_m⊗], rnd

Operation
IF (MP = 1) THEN
(tmp) ← ((IDX_i))*((Rw_m)) << 2

(ACC) ← (-ACC) + (tmp) + 0000 8000_H
ELSE
(tmp) ← $((IDX_i))*((Rw_m))$
(ACC) ← (-ACC) + (tmp) + 0000 8000_H
END IF
(MAL) ← 0 $((IDX_i(-\otimes))) \leftarrow ((IDX_i))$

Data Types

DOUBLE WORD

Result

40-bit signed value

Description

Multiplies the two signed 16-bit source operands pointed to by IDX_i (i=0,1) and Rw_m (m=0-15), respectively. The obtained signed 32-bit product is first sign-extended, then and on condition the MP flag is set, it is one-bit left 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_i overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX_i, as explained by the following table

Addressing Mode	Overwritten Address
$[IDX_i]$	(no change)
$[IDX_i+]$	(IDX_i) - 2
$[IDX_i$ -]	(IDX_i) + 2
$[IDX_i+QX_j]$	(IDX_i) - (QX_j)
$[IDX_i$ -QXj]	(IDX_i) + (QX_j)

MAC Condition Flags

N	Z	C	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.

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$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm D8 rrrr:qqqq	4
CoMACM-	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm E8 rrrr:qqqq	4
CoMACM	[IDX$_i$⊗], [Rw$_m$⊗], rnd	Yes	93 Xm D9 rrrr:qqqq	4
CoMACMR	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm F8 rrrr:qqqq	4
CoMACMR	[IDX$_i$⊗], [Rw$_m$⊗], rnd	Yes	93 Xm F9 rrrr:qqqq	4

Examples

```
CoMACM    [IDX1+QX0],[R1+QR1], rnd    ; (ACC)←(ACC)+((IDX0))*((R10))+rnd
                                      ; (R10) ← (R10) + (QR1)
                                      ; ( ((IDX1)-(QX0)) ) ← ((IDX1))
                                      ; (IDX1) ← (IDX1) + (QX0)
Repeat 3 times CoMACM [IDX0 - QX0], [R8+QR0]
                                      ; (ACC) ← (ACC) + ((IDX0))*((R8]
                                      ; (R8) ← (R8) + (QR0)
                                      ; ( ((IDX0) + (QX0)) ) ← ((IDX0))
                                      ; (IDX0) ← (IDX0) - (QX0)
Repeat MRW times CoMACM- [IDX1+QX1], [R7 - QR0]
                                      ; (ACC) ← (ACC) - ((IDX1))*((R7))
                                      ; (R7) ← (R7) - (QR0)
                                      ; ( ((IDX1) - (QX1)) ) ← ((IDX1))
                                      ; (IDX1) ← (IDX1) + (QX1)
```

CoMACM(R)u(-) Unsign. Multiply-Accumulate Parallel Data Move & Optional Round

Group Multiply/Multiply-Accumulate Instructions

Syntax CoMACMu [$IDX_i\otimes$], [$Rw_m\otimes$]

Operation
(tmp) $\leftarrow$ ((IDX_i))*((Rw_m))
(ACC) $\leftarrow$ (ACC) + (tmp) ((IDX_i(-$\otimes$))) $\leftarrow$ ((IDX_i))

Syntax CoMACMu [$IDX_i\otimes$], [$Rw_m\otimes$], rnd

Operation
(tmp) $\leftarrow$ ((IDX_i))*((Rw_m))
(ACC) $\leftarrow$ (ACC) + (tmp) + $0000\ 8000_H$
(MAL) $\leftarrow$ 0 ((IDX_i(-$\otimes$))) $\leftarrow$ ((IDX_i))

Syntax CoMACMu- [$IDX_i\otimes$], [$Rw_m\otimes$]

Operation
(tmp) $\leftarrow$ ((IDX_i))*((Rw_m))
(ACC) $\leftarrow$ (ACC) - (tmp) ((IDX_i(-$\otimes$))) $\leftarrow$ ((IDX_i))

Syntax CoMACMRu [$IDX_i\otimes$], [$Rw_m\otimes$]

Operation
(tmp) $\leftarrow$ ((IDX_i))*((Rw_m))
(ACC) $\leftarrow$ (tmp) - (ACC) ((IDX_i(-$\otimes$))) $\leftarrow$ ((IDX_i))

Syntax CoMACMRu [$IDX_i\otimes$], [$Rw_m\otimes$], rnd

Operation
(tmp) $\leftarrow$ ((IDX_i))*((Rw_m))
(ACC) $\leftarrow$ (tmp) - (ACC) + $0000\ 8000_H$
(MAL) $\leftarrow$ 0 ((IDX_i(-$\otimes$))) $\leftarrow$ ((IDX_i))

Data Types DOUBLE WORD

Result 40-bit unsigned value

Description Multiplies the two unsigned 16-bit source operands pointed to by IDX_i (i=0,1) and Rw_m (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_i overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX_i, as illustrated by the following table

Addressing Mode	Overwritten Address
[IDX_i]	(no change)
[IDX_i+]	(IDX_i)- 2
[IDX_i -]	(IDX_i) + 2
[IDX_i+QX_j]	(IDX_i) - (QX_j)
[IDX_i -QXj]	(IDX_i) + (QX_j)

MAC Condition Flags

N	Z	C	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.

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
CoMACMu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 18 rrrr:rqqq	4
CoMACMu-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 28 rrrr:rqqq	4
CoMACMu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 19 rrrr:rqqq	4
CoMACMRu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 38 rrrr:rqqq	4
CoMACMRu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 39 rrrr:rqqq	4

Examples

```
CoMACMu   [IDX1+QX0], [R10+QR1], rnd   ; (ACC)←(ACC) + ((IDX0)) * ((R10))+ rnd
                                       ; (R10) ← (R10) + (QR1)
                                       ; [IDX1-2.QX0] ← [IDX1]
                                       ; (IDX1) ← (IDX1) + (QX0)
Repeat 3 times CoMACMu [IDX0 - QX0], [R8+QR0]
                                       ; (ACC) ←(ACC) + ((IDX0))*((R8))
                                       ; (R8) ← (R8) + (QR0)
                                       ; ( ((IDX0) + (QX0)) ) ← ((IDX0))
                                       ; (IDX0) ←(IDX0) - (QX0)
Repeat MRW times CoMACMRu [IDX1+QX1], [R7 - QR0]
                                       ; (ACC) ←((IDX1))*((R7)) - (ACC)
                                       ; (R7) ← (R7) - (QR0)
                                       ; ( ((IDX1) - (QX1)) ) ←((IDX1))
                                       ; (IDX1) ← (IDX1) + (QX1)
```

CoMACM(R)su(-) Mix. Multiply-Accumulate Parallel Data Move and Optional Round

Group Multiply/Multiply-Accumulate Instructions

Syntax CoMACMsu [IDX_i⊗], [Rw_m⊗]

Operation (tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp) ((IDX_i(-⊗))) ← ((IDX_i))

Syntax CoMACMsu [IDX_i⊗], [Rw_m⊗], rnd

Operation (tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp) + 0000 8000_H
(MAL) ← 0 ((IDX_i(-⊗))) ← ((IDX_i))

Syntax CoMACMsu- [IDX_i⊗], [Rw_m⊗]

Operation (tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) - (tmp) ((IDX_i(-⊗))) ← ((IDX_i))

Syntax CoMACMRsu [IDX_i⊗], [Rw_m⊗]

Operation (tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (tmp) - (ACC) ((IDX_i(-⊗))) ← ((IDX_i))

Syntax CoMACMRsu [IDX_i⊗], [Rw_m⊗], rnd

Operation (tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (tmp) - (ACC) + 0000 8000_H
(MAL) ← 0 ((IDX_i(-⊗))) ← ((IDX_i))

Data Types DOUBLE WORD

Result 40-bit signed value

Description Multiplies the two signed and unsigned 16-bit source operands pointed to by IDX_i (i=0,1) and Rw_m (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_i overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX_i, as illustrated by the following table

Addressing Mode	Overwritten Address
[IDX_i]	(no change)
[IDX_i+]	(IDX_i) - 2
[IDX_i -]	(IDX_i) + 2
[IDX_i+QX_j]	(IDX_i) - (QX_j)
[IDX_i - QXj]	(IDX_i) + (QX_j)

MAC Condition Flags

N	Z	C	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.

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
CoMACMsu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 98 rrrr:rqqq	4
CoMACMsu-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm A8 rrrr:rqqq	4
CoMACMsu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 99 rrrr:rqqq	4
CoMACMRsu	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm B8 rrrr:rqqq	4
CoMACMRsu	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm B9 rrrr:rqqq	4

Example

```
CoMACMsu  [IDX1+QX0], [R10+QR1], rnd    ; (ACC) ←(ACC) + ((IDX1))*((R10)) + rnd
                                        ; (R10) ← (R10) + (QR1)
                                        ; ( ((IDX1) -(QX0)) ) ← ((IDX1))
                                        ; (IDX1) ← (IDX1) + (QX0)
Repeat 3 times CoMACMsu [IDX0 - QX0], [R8+QR0], rnd
                                        ; (ACC) ← (ACC) + ((IDX0))*((R8))
                                        ; (R8) ← (R8) + (QR0)
                                        ; ( ((IDX0) + (QX0)) )← ((IDX0))
                                        ; (IDX0) ← (IDX0) - (QX0)
Repeat MRW times CoMACMRsu [IDX1+QX1], [R7 - QR0], rnd
                                        ; (ACC) ← ((IDX1))*((R7)) - (ACC) + rnd
                                        ; (R7) ← (R7) - (QR0)
                                        ; ( ((IDX1)) - (QX1)) ) ← ((IDX1))
                                        ; (IDX1) ← (IDX1) + (QX1)
```

CoMACM(R)us(-) Mix. Multiply-Accumulate Parallel Data Move and Optional Round

Group Multiply/Multiply-Accumulate Instructions

Syntax CoMACMus [IDX_i⊗], [Rw_m⊗]

Operation
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp) ((IDX_i(-⊗))) ← ((IDX_j))

Syntax CoMACMus [IDX_i⊗], [Rw_m⊗], rnd

Operation
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) + (tmp) + 0000 8000_H
(MAL) ← 0 ((IDX_i(-⊗))) ← ((IDX_j))

Syntax CoMACMus- [IDX_i⊗], [Rw_m⊗]

Operation
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (ACC) - (tmp) ((IDX_i(-⊗))) ← ((IDX_j))

Syntax CoMACMRus [IDX_i⊗], [Rw_m⊗]

Operation
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (tmp) - (ACC) ((IDX_i(-⊗))) ← ((IDX_j))

Syntax CoMACMRus [IDX_i⊗], [Rw_m⊗], rnd

Operation
(tmp) ← ((IDX_i))*((Rw_m))
(ACC) ← (tmp) - (ACC) + 0000 8000_H
(MAL) ← 0 ((IDX_i(-⊗))) ← ((IDX_j))

Data Types DOUBLE WORD

Result 40-bit signed value

Description

Multiplies the two unsigned and signed 16-bit source operands pointed to by IDX_i (i=0,1) and Rw_m (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_i overwrites another data located in memory (DPRAM). The address of the overwritten data depends on the operation executed on IDX_i, as illustrated by the following table

Addressing Mode	Overwritten Address
[IDX_i]	(no change)
[IDX_i+]	(IDX_i) - 2
[IDX_i -]	(IDX_i) + 2
[IDX_i+QX_j]	(IDX_i) - (QX_j)
[IDX_i - QXj]	(IDX_i) + (QX_j)

MAC Condition Flags

N	Z	C	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.

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
CoMACMus	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 58 rrrr:rqqq	4
CoMACMus-	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 68 rrrr:rqqq	4
CoMACMus	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 59 rrrr:rqqq	4
CoMACMRus	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 78 rrrr:rqqq	4
CoMACMRus	[IDX_i⊗], [Rw_m⊗], rnd	Yes	93 Xm 79 rrrr:rqqq	4

Examples

```
CoMACMus  [IDX1+QX0], [R10+QR1], rnd   ; (ACC) ← (ACC) + ((IDX0))*((R10)) +rnd
                                       ; (R11) ← (R11) + (QR1)
                                       ; ( ((IDX1) - (QX0)) )← ((IDX1))
                                       ; (IDX1) ← (IDX1) + (QX0)
Repeat 3 times CoMACMus [IDX0 - QX0], [R8+QR0]
                                       ; (ACC) ← (ACC) + ((IDX0))*((R8))
                                       ; (R8) ← (R8) + (QR0)
                                       ; ( ((IDX0) + (QX0)) ) ← ((IDX0))
                                       ; (IDX0) ← (IDX0) - (QX0)
Repeat MRW times CoMACMRus [IDX1+QX1], [R7 - QR0], rnd
                                       ; (ACC) ← ((IDX1))*((R7)) - (ACC) + rnd
                                       ; (R7) ← (R7) - (QR0)
                                       ; ( ((IDX1) - (QX1)) )← ((IDX1))
                                       ; (IDX1) ← (IDX1) + (QX1)
```

CoADD(2) **Add**

Group
32-bit Arithmetic Instructions

Syntax
CoADD op1, op2

Operation
(tmp) ← (op2)\(op1)
(ACC) ← (ACC) + (tmp)

Syntax
CoADD2 op1, op2

Operation
(tmp) ← 2 * (op2)\(op1)
(ACC) ← (ACC) + (tmp)

Data Types
DOUBLE WORD

Result
40-bit signed value

Description
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 $ffff_h$ or ff 8000 0000_h, respectively. This instruction is repeatable with indirect addressing modes and allows up to two parallel memory reads

MAC Condition Flags

N	Z	C	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.

C Set if a carry is generated. Cleared otherwise.

SV Set if an arithmetic overflow occurred. Not affected otherwise.

E Set if MAE is used. Cleared otherwise.

SL Set if the contents of the ACC is automatically saturated. Not affected otherwise.

Note
The E-flag is set when the nine highest bits of the accumulator are not equal.
The SV-flag is set, when a 40-bit arithmetic overflow/ underflow occurs.

Addressing Modes

Mnemonic		Repeatable	Format	Bytes
CoADD	Rw_n, Rw_m	No	A3 nm 02 00	4
CoADD2	Rw_n, Rw_m	No	A3 nm 42 00	4
CoADD	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 02 rrrr:rqqq	4
CoADD2	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 42 rrrr:rqqq	4
CoADD	Rw_n, [Rw_m⊗]	Yes	83 nm 02 rrrr:rqqq	4
CoADD2	Rw_n, [Rw_m⊗]	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 times CoADD [IDX1+QX1], [R10+QR0]
                                           ; (ACC) ← (ACC) + ( ((R10))\((IDX1)) )
                                           ; (R10) ← (R10) + (QR0)
                                           ; (IDX1) ← (IDX1) + (QX1)
Repeat MRW times CoADD2 R4, [R8 - QR1]     ; (ACC) ← (ACC) + 2*( ((R8))\(R4) )
                                           ; (R8) ← (R8) - (QR1)
```

Addition Examples

Instr.	MS	op 1	op 2	ACC (before)	ACC (after)	N	Z	C	SV	E	SL
CoADD	x	0000_H	$FFFF_H$	00 0100 0000_H	00 00FF 0000_H	0	0	1	-	0	-
CoADD2	x	0000_H	0200_H	00 0300 0000_H	00 0700 0000_H	0	0	0	-	0	-
CoADD	0	0000_H	4000_H	7F BFFF $FFFF_H$	7F FFFF $FFFF_H$	0	0	0	-	1	-
CoADD	0	0001_H	4000_H	7F BFFF $FFFF_H$	80 0000 0000_H	1	0	0	1	1	-
CoADD	0	$FFFF_H$	$FFFF_H$	FF FFFF $FFFF_H$	FF FFFF $FFFE_H$	1	0	1	-	0	-
CoADD	0	$FFFF_H$	$FFFF_H$	00 0000 0001_H	00 0000 0000_H	0	1	1	-	0	-
CoADD	0	$FFFF_H$	$FFFF_H$	80 0000 0000_H	7F FFFF $FFFF_H$	0	0	1	1	1	-
CoADD2	0	0001_H	2000_H	FF C000 0001_H	00 0000 0003_H	0	0	1	-	0	-
CoADD2	0	0001_H	1800_H	FF C000 0001_H	FF F000 0003_H	1	0	0	-	0	-
CoADD	0	$B4A1_H$	$73C2_H$	00 7241 $A0C3_H$	00 E604 5564_H	0	0	0	-	1	-
	1				00 7FFF $FFFF_H$	0	0	0	-	0	1
CoADD	0	$B4A1_H$	$A3C2_H$	FF 8241 $A0C3_H$	FF 2604 5564_H	1	0	1	-	1	-
	1				FF 8000 0000_H	1	0	1	-	0	1
CoADD	0	$B4A1_H$	$73C2_H$	7F B241 $A0C3_H$	80 2604 5564_H	1	0	0	1	1	-
CoADD	0	$B4A1_H$	$A3C2_H$	80 0241 $A0C3_H$	7F A604 5564_H	0	0	1	1	1	-

CoSUB(2)(R) Subtract

Group
32-bit Arithmetic Instructions

Syntax
CoSUB op1, op2

Operation
(tmp) $\leftarrow$ (op2)\(op1)
(ACC) $\leftarrow$ (ACC) - (tmp)

Syntax
CoSUB2 op1, op2

Operation
(tmp) $\leftarrow$ 2 * (op2)\(op1)
(ACC) $\leftarrow$ (ACC) - (tmp)

Syntax
CoSUBR op1, op2

Operation
(tmp) $\leftarrow$ (op2)\(op1)
(ACC) $\leftarrow$ (tmp) - (ACC)

Syntax
CoSUB2R op1, op2

Operation
(tmp) $\leftarrow$ 2 * (op2)\(op1)
(ACC) $\leftarrow$ (tmp) - (ACC)

Data Types
DOUBLE WORD

Result
40-bit signed value

Description
Subtracts a 40-bit operand from the 40-bit Accumulator contents or vice versa when “r” option is used 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 subtracted/added from/to ACC/negated 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$_h$ or ff 8000 0000$_h$, respectively. This instruction is repeatable with indirect addressing modes and allows up to two parallel memory reads

MAC Condition Flags

N	Z	C	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.

C Set if a 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.

Note
The E-flag is set when the nine highest bits of the accumulator are not equal.
The SV-flag is set, when a 40-bit arithmetic overflow/ underflow occurs.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoSUB	Rw$_n$, Rw$_m$	No	A3 nm 0A 00	4
CoSUBR	Rw$_n$, Rw$_m$	No	A3 nm 12 00	4
CoSUB2	Rw$_n$, Rw$_m$	No	A3 nm 4A 00	4
CoSUB2R	Rw$_n$, Rw$_m$	No	A3 nm 52 00	4
CoSUB	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm 0A rrrr:rqqq	4
CoSUBR	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm 12 rrrr:rqqq	4
CoSUB2	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm 4A rrrr:rqqq	4
CoSUB2R	[IDX$_i$⊗], [Rw$_m$⊗]	Yes	93 Xm 52 rrrr:rqqq	4
CoSUB	Rw$_n$, [Rw$_m$⊗]	Yes	83 nm 0A rrrr:rqqq	4
CoSUBR	Rw$_n$, [Rw$_m$⊗]	Yes	83 nm 12 rrrr:rqqq	4
CoSUB2	Rw$_n$, [Rw$_m$⊗]	Yes	83 nm 4A rrrr:rqqq	4
CoSUB2R	Rw$_n$, [Rw$_m$⊗]	Yes	83 nm 52 rrrr:rqqq	4

Examples

CoSUB	R0, R1	; (ACC) ← (ACC) - (R1)\(R0)
CoSUB2	R2, [R6+]	; (ACC) ← (ACC) + 2*(((R6))\ (R2))
		; (R6) ← (R6) + 2
Repeat 3 timesCoSUB [IDX1+QX1], [R10+QR0]		
		; (ACC) ← (ACC) - (((R10))\((IDX1)))
		; (R10) ← (R10) + (QR0)
		; (IDX1) ← (IDX1) + (QX1)
Repeat MRW timesCoSUBr2 R4, [R8 - QR1]		; (ACC) ← 2*(((R8))\(R4))- (ACC)
		; (R8) ← (R8) - (QR1)

Subtraction Examples

Instr.	MS	op 1	op 2	ACC (before)	ACC (after)	N	Z	C	SV	E	SL
CoSUB	x	183A$_H$	72AC$_H$	00 7FFF FFFF$_H$	00 0D53 E7C5$_H$	0	0	0	-	0	-
CoSUBr	x	183A$_H$	72AC$_H$	00 7FFF FFFF$_H$	FF F2AC 183B$_H$	1	0	1	-	0	-
CoSUB2	x	0C1D$_H$	3956$_H$	00 E604 5564$_H$	00 7358 3D2A$_H$	0	0	0	-	0	-
CoSUBr2	x	0C1D$_H$	3956$_H$	00 E604 5564$_H$	FF 8CA7 C2D6$_H$	1	0	1	-	0	-
CoSUB	0	FFFF$_H$	FFFF$_H$	7F FFFF FFFF$_H$	80 0000 0000$_H$	1	0	1	1	1	-
	1				00 7FFF FFFF$_H$	0	0	1	1	0	1
CoSUB2	0	0000$_H$	3000$_H$	7F FFFF FFFF$_H$	7F 9FFF FFFF$_H$	0	0	0	-	1	-
CoSUB2	0	0001$_H$	0000$_H$	80 0000 0000$_H$	7F FFFF FFFE$_H$	0	0	0	1	1	-
	1				FF 8000 0000$_H$	1	0	0	1	0	1

CoNEG

Negate Accumulator with Optional Rounding

Group

32-bit Arithmetic Instructions

Syntax

CoNEG
CoNEG rnd

Operation

IF (rnd) THEN
(ACC) ← 0 - (ACC) + 00008000_H
(MAL) ← 0
ELSE
(ACC) ← 0 - (ACC)
END IF

Data Types

ACCUMULATOR

Result

40-bit signed value

Description

The Accumulator content is subtracted from zero and the result is optionally rounded before being stored in the accumulator register. with “rnd” option 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 $ffff_h$ or ff 8000 0000_h, respectively. This instruction is not repeatable

MAC Condition Flags

N	Z	C	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.

C Set if a 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
CoNEG	No	A3 00 32 00	4
CoNEG rnd	No	A3 00 72 00	4

Examples

```
CoNEG            ; (ACC) ← 0 - (ACC)
CoNEG    rnd     ; (ACC) ← 0 - (ACC) + rnd
```

Instr.	MS	rnd	ACC (before)	ACC (after)	N	Z	C	SV	E	SL
CoNEG	x	No	00 1234 5678_H	FF EDCB $A988_H$	1	0	0	-	0	-
CoNEG	x	Yes	00 1234 5678_H	FF EDCC 0000_H	1	0	0	-	0	-

CoABS Absolute Value

Group 32-bit Arithmetic Instructions

Syntax CoABS

Operation (ACC) ← Abs(ACC)

Syntax CoABS op1, op2

Operation (ACC) ← Abs((op2)\(op1))

Data Types ACCUMULATOR, DOUBLE WORD

Result 40-bit signed value

Description

Compute the absolute value of the Accumulator if no operands are specified or the absolute value of a 40-bit source operand and load 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. This instruction is not repeatable

MAC Condition Flags

N	Z	C	SV	E	SL
*	*	0	-	*	*

N	Set if the most significant bit of the result is set. Cleared otherwise.
Z	Set if the result equals zero. Cleared otherwise.
C	Always cleared.
SV	Not affected.
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
CoABS		No	A3 00 1A 00	4
CoABS	Rw_n, Rw_m	No	A3 nm CA 00	4
CoABS	[IDX_i⊗], [Rw_m⊗]	No	93 Xm CA 0:0qqq	4
CoABS	Rw_n, [Rw_m⊗]	No	83 nm CA 0:0qqq	4

CoLOAD(2)(-) Load Accumulator

Group 32-bit Arithmetic Instructions

Syntax CoLOAD op1, op2

Operation
(tmp) ← (op2)\(op1)
(ACC) ← 0 + (tmp)

Syntax CoLOAD- op1, op2

Operation
(tmp) ← (op2)\(op1)
(ACC) ← 0 - (tmp)

Syntax CoLOAD2 op1, op2

Operation
(tmp) ← 2*((op2)\(op1))
(ACC) ← 0 + (tmp)

Syntax CoLOAD2- op1, op2

Operation
(tmp) ← 2*((op2)\(op1))
(ACC) ← 0 - (tmp)

Data Types DOUBLE WORD

Result 40-bit signed value

Description Loads the accumulator with a 40-bit source operand. The 40-bit source operand results from the concatenation of the two source operands op1 (LSW) and op2 (MSW) which is then sign-extended. "**2**" and "-" options indicate that the 40-bit operand is also multiplied by two or/and negated, respectively, prior being stored in the accumulator. The "-" option indicates that the source operand is 2's complemented. 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 $ffff_h$ or ff 8000 0000_h, respectively. This instruction is not repeatable and allows up to two parallel memory reads.

MAC Condition Flags

N	Z	C	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.

C Set if a borrow is generated. Cleared otherwise.

SV Not affected.

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
CoLOAD	Rw_n, Rw_m	No	A3 nm 22 00	4
CoLOAD-	Rw_n, Rw_m	No	A3 nm 2A 00	4
CoLOAD2	Rw_n, Rw_m	No	A3 nm 62 00	4
CoLOAD2-	Rw_n, Rw_m	No	A3 nm 6A 00	4
CoLOAD	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 22 0:0qqq	4
CoLOAD-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 2A 0:0qqq	4
CoLOAD2	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 62 0:0qqq	4
CoLOAD2-	[IDX_i⊗], [Rw_m⊗]	No	93 Xm 6A 0:0qqq	4
CoLOAD	Rw_n, [Rw_m⊗]	No	83 nm 22 0:0qqq	4
CoLOAD-	Rw_n, [Rw_m⊗]	No	83 nm 2A 0:0qqq	4
CoLOAD2	Rw_n, [Rw_m⊗]	No	83 nm 62 0:0qqq	4
CoLOAD2-	Rw_n, [Rw_m⊗]	No	83 nm 6A 0:0qqq	4

CoNOP — No-Operation

Group

32-bit Arithmetic Instructions

Syntax

CoNOP

Operation

No Operation

Description

Modifies the address pointers without changing the internal MAC-Unit registers.

MAC Condition Flags

N	Z	C	SV	E	SL
-	-	-	-	-	-

N	Not affected.
Z	Not affected.
C	Not affected.
SV	Not affected.
E	Not affected.
SL	Not affected.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoNOP	[Rw_m⊗]	Yes	93 0m 5A rrrr:rqqq	4
CoNOP	[IDX_i⊗]	Yes	93 X0 5A rrrr:r000	4
CoNOP	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 5A rrrr:rqqq	4

Examples

```
CoNOP      [IDX0+QX1], [R11+QR1]       ; (R11) ← (R11) + (QR1)
                                       ; (IDX0) ← (IDX0) + (QX1)
CoNOP      [R1+]                       ; (R1) ← (R1) + 2
Repeat MRW times CoNOP [IDX1+QX0]      ; (IDX1) ← (IDX1) + (QR0)
```

CoSHL — Accumulator Logical Shift Left

Group
Shift Instructions

Syntax
CoSHL op1

Operation
(count) ← (op1)
(C) ← 0
DO WHILE (count) ≠ 0
(C) ← (ACC39)
(ACCn) ← (ACCn-1) [n=1...39]
(ACC0) ← 0
(count) ← (count) -1
END WHILE

Data Types
ACCUMULATOR

Result
40-bit signed value

Description
Shifts the ACC register left by as many times as specified by the operand op1. The least significant bits of the result are filled with zeros. Only shift values contained between 0 and 8 are allowed. "op1" can be either a 4-bit unsigned immediate data, or the least significant 4 bits (considered as unsigned data) of any register directly or indirectly addressed operand. 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 $ffff_h$ or ff 8000 0000_h, resp. This instruction is repeatable when "op 1" is not an immediate operand.

MAC Condition Flags

N	Z	C	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.
C The carry flag is set according to the last MSB shifted out of op1.
SV Set if the last shifted out bit is different from N.
E Set if the MAE is used. Cleared otherwise.
SL Set if the content of the ACC is automatically saturated. Not affected otherwise.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoSHL	Rw_n	Yes	A3 nn 8A rrrr:r000	4
CoSHL	$\#data_4$	No	A3 00 82 ssss:0	4
CoSHL	$[Rw_m\otimes]$	Yes	83 mm 8A rrrr:rqqq	4

Examples

```
CoSHL   #3            ; (ACC) ← (ACC) << 3
CoSHL   R3            ; (ACC) ← (ACC) << (R33-0)
CoSHL   [R10 - QR0]   ; (ACC) ← (ACC) << ((R103-0))
                      ; (R10) ← (R10) - (QR0)
```

CoSHR

Accumulator Logical Shift Right

Group

Shift Instructions

Syntax

CoSHR op1

Operation

(count) ← (op1)
(C) ← 0
DO WHILE (count) ≠ 0
((ACCn) ← (ACCn+1) [n=0...38]
(ACC39) ← 0
(count) ← (count) -1
END WHILE

Data Types

ACCUMULATOR

Result

40-bit signed value

Description

Shifts the ACC register right by as many times as specified by the operand op1. The most significant bits of the result are filled with zeros accordingly. Only shift values contained between 0 and 8 are allowed. “op1” can be either a 4-bit unsigned immediate data, or the least significant 4 bits (considered as unsigned data) of any register directly or indirectly addressed operand. The MS bit of the MCW register does not affect the result. This instruction is repeatable when “op 1” is not an immediate operand.

MAC Condition Flags

N	Z	C	SV	E	SL
*	*	0	-	*	-

N	Set if the most significant bit of the result is set. Cleared otherwise.
Z	Set if the result equals zero. Cleared otherwise.
C	Cleared always.
SV	Not affected.
E	Set if the MAE is used. Cleared otherwise.
SL	Not affected.

Addressing Modes

Mnemonic		Repeatable	Format	Bytes
CoSHR	Rw_n	Yes	A3 nn 9A rrrr:r000	4
CoSHR	$\#data_4$	No	A3 00 92 ssss:0	4
CoSHR	$[Rw_m\otimes]$	Yes	83 mm 9A rrrr:rqqq	4

Examples

```
CoSHR   #3              ; (ACC) ← (ACC) >> 3
CoSHR   R3              ; (ACC) ← (ACC) >> (R33-0)
CoSHR   [R10 - QR0]     ; (ACC) ← (ACC) >> ((R103-0))
                        ; (R10) ← (R10) - (QR0)
```

CoASHR

Accumulator Arithmetic Shift Right with Optional Round

Group

Shift Instructions

Syntax

CoASHR op1
CoASHR op1, rnd

Operation

(count) $\leftarrow$ (op1)
(C) $\leftarrow$ 0
DO WHILE (count) $\neq$ 0
(ACCn) $\leftarrow$ (ACCn+1) [n=0...38]
(count) $\leftarrow$ (count) -1
END WHILE
IF (rnd) THEN
(ACC) $\leftarrow$ (ACC) + 00008000_H
(MAL) $\leftarrow$ 0
END IF

Data Types

ACCUMULATOR

Result

40-bit signed value

Description

Arithmetically shifts the ACC register right by as many times as specified by the operand op1. To preserve the sign of the ACC register, the most significant bits of the result are filled with sign 0 if the original MSB was a 0 or with ones if the original MSB was 1. Only shift values between 0 and 8 are allowed. “op1” can be either a 4-bit unsigned immediate data, or the least significant 4 bits (considered as unsigned data) of any register directly or indirectly addressed operand. Without “rnd” option, the MS bit of the MCW register does not affect the result. While with “rnd” option and if the MS bit is set and when a 32-bit overflow or underflow occurs, the obtained result becomes 00 7ffff ffff_h or ff 8000 0000_h, respectively. This instruction is repeatable when “op 1” is not an immediate operand.

MAC Condition Flags

N	Z	C	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.

C Set if a carry is generated (rnd). Cleared otherwise.

SV Set if an arithmetic overflow occurred (rnd). Not affected otherwise.

E Set if the MAE is used. Cleared otherwise.

SL Set if the contents of the ACC is automatically saturated (rnd). Not affected otherwise

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoASHR	Rw_n	Yes	A3 nn AA rrrr:r000	4
CoASHR	Rw_n, rnd	Yes	A3 nn BA rrrr:r000	4
CoASHR	$\#data_4$	No	A3 00 A2 ssss:0	4
CoASHR	$\#data_4$, rnd	No	A3 00 B2 ssss:0	4
CoASHR	$[Rw_m\otimes]$	Yes	83 mm AA rrrr:rqqq	4
CoASHR	$[Rw_m\otimes]$, rnd	Yes	83 mm BA rrrr:rqqq	4

Examples

```
CoASHR   #3, rnd        ; (ACC) ← (ACC) >>a 3 + rnd
CoASHR   R3             ; (ACC) ← (ACC) >>a (R33-0)
CoASHR   [R10 - QR0]    ; (ACC) ← (ACC) >>a ((R103-0))
                        ; (R10) ← (R10) - (QR0)
```

CoRND

Round Accumulator

Group
Shift Instructions

Syntax
CoRND

Operation
(ACC) ← (ACC) + 0000 8000_H
(MAL) ← 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 $ffff_h$ or ff 800 0000_h, respectively. This instruction is not repeatable.

MAC Condition Flags

N	Z	C	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.
C	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.

Addressing Modes

Mnemonic	Rep	Format	Bytes
CoRND	No	A3 00 B2 00	4

Notes
CoRND is equivalent to CoASHR #0, rnd.

Example

```
CoRND                                   ; (ACC) ← (ACC) + rnd
```

CoMAX

Maximum

Group

Compare Instructions

Syntax

CoMAX op1 op2

Operation

(tmp) ← (op2)\(op1)
(ACC) ← 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
*	*	0	-	*	*

N	Set if the most significant bit of the result is set. Cleared otherwise.
Z	Set if the result equals zero. Cleared otherwise.
C	Cleared always.
SV	Not affected.
E	Set if the MAE is used. Cleared otherwise.
SL	Set if the contents of the ACC register is changed. Not affected otherwise.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoMAX	Rw_n, Rw_m	No	A3 nm 3A 00	4
CoMAX	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 3A rrrr:rqqq	4
CoMAX	Rw_n, [Rw_m⊗]	Yes	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) ← Max( (ACC), ((R6))\(R5)) )
                                    ; (R6) ← (R6) - (QR0)
```

CoMIN Minimum

Group Compare Instructions

Syntax CoMIN op1, op2

Operation

(tmp) ← (op2)\(op1)
(ACC) ← 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
*	*	0	-	*	*

N	Set if the most significant bit of the result is set. Cleared otherwise.
Z	Set if the result equals zero. Cleared otherwise.
C	Cleared always.
SV	Not affected.
E	Set if the MAE is used. Cleared otherwise.
SL	Set if the contents of the ACC register is changed. Not affected otherwise.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoMIN	Rw_n, Rw_m	No	A3 nm 7A 00	4
CoMIN	[IDX_i⊗], [Rw_m⊗]	Yes	93 Xm 7A rrrr:rqqq	4
CoMIN	Rw_n, [Rw_m⊗]	Yes	83 nm 7A rrrr:rqqq	4

Examples

```
CoMIN    [IDX1+QX0], [R11+QR1]          ; (ACC) ← min( (ACC), ((R11))\((IDX1)) )
                                        ; (R11) ← (R11) + (QR1)
                                        ; (IDX1) ← (IDX1) + (QX0)
CoMIN    R1, R10                        ; (ACC) ← min( (ACC), (R10)\(R1) )
Repeat 23 times CoMIN R5, [R6 - QR0]    ; (ACC) ← min( (ACC), ((R6))\(R5)) )
                                        ; (R6) ← (R6) - (QR0)
```

CoCMP

Compare

Group

Compare Instructions

Syntax

CoCMP op1, op2

Operation

tmp ← (op2)\(op1)
(ACC) ⇔ (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 sign-extended. 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	C	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.

C Set if a borrow is generated. Cleared otherwise.

SV Not affected.

E Not affected.

SL Not affected.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoCMP	Rw_n, Rw_m	No	A3 nm C2 00	4
CoCMP	[IDX_i⊗], [Rw_m⊗]	No	93 Xm C2 0:0qqq	4
CoCMP	Rw_n, [Rw_m⊗]	No	83 nm C2 0:0qqq	4

Examples

```
CoCMP   [IDX1+QX0], [R11+QR1]   ; MSW(N,Z,C) ←(ACC) - ((R11))\((IDX1))
                                ; (R11) ← (R11) + (QR1)
                                ; (IDX1) ← (IDX1) + (QX0)
CoCMP   R1, [R2-]               ; MSW(N,Z,C) ← (ACC) - ((R2))\(R1)
                                ; (R2) ← (R2) - 2
CoCMP   R2, R5                  ; MSW(N,Z,C) ← (ACC) - (R5)\(R2)
```

CoSTORE

Store a MAC-Unit Register

Group

Transfer Instructions

Syntax

CoSTORE 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	C	SV	E	SL
-	-	-	-	-	-

N	Not affected
Z	Not affected
C	Not affected
SV	Not affected
E	Not affected
SL	Not affected

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoSTORE	Rw_n, CoReg	No	C3 nn wwww:w000 00	4
CoSTORE	[Rw_n⊕], CoReg	Yes	B3 nn wwww:w000 rrrr:rqqq	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    [R11+QR1], MAS              ; ((R11)) ← limited((ACC))
                                       ; (R11) ←(R11) + (QR1)
Repeat 3 times CoSTORE [R2-], MAL      ; ((R2)) ← (MAL)
                                       ; (R2) ← (R2) - 2
```

CoMOV

Memory to Memory Move

Group

Transfer Instructions

Syntax

CoMOV op1, op2

Operation

(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 Flags

E	Z	V	C	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 the value of the source operand op2 equals zero. Cleared otherwise.

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	C	SV	E	SL
-	-	-	-	-	-

N Not affected.

Z Not affected.

C Not affected.

SV Not affected.

E Not affected.

SL Not affected.

Addressing Modes

Mnemonic		Rep	Format	Bytes
CoMOV	$[IDX_i\otimes]$, $[Rw_m\otimes]$	Yes	D3 Xm 00 rrrr:rqqq	4

Examples

Repeat 24 times CoMOV [IDX1+QX0], [R11+QR1]

; ((IDX1)) ← ((R11))
; (R11) ← (R11) + (QR1)
; (IDX1) ← (IDX1) + (QX0)

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.