
ST10

FAMILY PROGRAMMING MANUAL

Release 1



Ref: ST10FPM

TABLE OF CONTENTS		Page
1	INTRODUCTION	3
2	STANDARD INSTRUCTION SET	4
2.1	ADDRESSING MODES.....	4
2.1.1	Short addressing modes.....	4
2.1.2	Long addressing mode.....	5
2.1.3	DPP override mechanism.....	6
2.1.4	Indirect addressing modes	6
2.1.5	Constants	7
2.1.6	Branch target addressing modes.....	7
2.2	INSTRUCTION EXECUTION TIMES	8
2.2.1	Definition of measurement units.....	9
2.2.2	Minimum state times.....	10
2.2.3	Additional state times	10
2.3	INSTRUCTION SET SUMMARY.....	13
2.4	INSTRUCTION SET ORDERED BY FUNCTIONAL GROUP	15
2.5	INSTRUCTION SET ORDERED BY OPCODES	26
2.6	INSTRUCTION CONVENTIONS.....	34
2.6.1	Instruction name	34
2.6.2	Syntax.....	34
2.6.3	Operation.....	34
2.6.4	Data types	35
2.6.5	Description.....	35
2.6.6	Condition code.....	35
2.6.7	Flags.....	36
2.6.8	Addressing modes.....	37
2.7	ATOMIC AND EXTENDED INSTRUCTIONS	38
2.8	INSTRUCTION DESCRIPTIONS	39
3	MAC INSTRUCTION SET	123
3.1	ADDRESSING MODES.....	123
3.2	MAC INSTRUCTION EXECUTION TIME	124
3.3	MAC INSTRUCTION SET SUMMARY.....	124
3.4	MAC INSTRUCTION CONVENTIONS.....	126
3.4.1	Operands.....	126
3.4.2	Operations	126
3.4.3	Abbreviations	126
3.4.4	Data addressing Modes.....	126
3.4.5	Instruction format.....	127
3.4.6	Flag states	127
3.4.7	Repeated instruction syntax	127
3.4.8	Shift value.....	127
3.5	MAC INSTRUCTION DESCRIPTIONS	127
4	REVISION HISTORY	170



ST10 FAMILY PROGRAMMING MANUAL

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. The MAC instructions are only available to devices containing the MAC, refer to the datasheet for device-specific information.

In the standard instruction set, addressing modes, 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, are provided for the optimization of instruction sequences.

Instruction set tables ordered by functional group, can be used to identify the best instruction for a given application. Instruction set tables 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, using the conventions defined in this manual. For ease of use, the instructions are listed alphabetically.

The MAC instruction set is divided into its 5 functional groups: Multiply and Multiply-Accumulate, 32-Bit Arithmetic, Shift, Compare and Transfer Instructions. Two new addressing modes supply the MAC with up to 2 new operands per instruction.

Cross reference tables of MAC instruction mnemonics by address mode, and MAC instruction mnemonic by functional code can be used for quick reference.

As for the standard instruction set, each instruction has been described individually in a standard format according to defined conventions. For convenience, the instructions are described in alphabetical order.

2 - STANDARD INSTRUCTION SET

2.1 - Addressing Modes

2.1.1 - Short 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. Short addressing modes use an implicit base offset address to specify the 24-bit physical address. Short addressing modes give access to the GPR, SFR or bit-addressable memory space $PhysicalAddress = BaseAddress + \Delta \times ShortAddress$.

Note: $\Delta = 1$ for byte GPRs, $\Delta = 2$ for word GPRs (see Table 1).

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. 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. 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.

Table 1 : Short addressing mode summary

Mnemo	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

2.1.2 - Long addressing mode

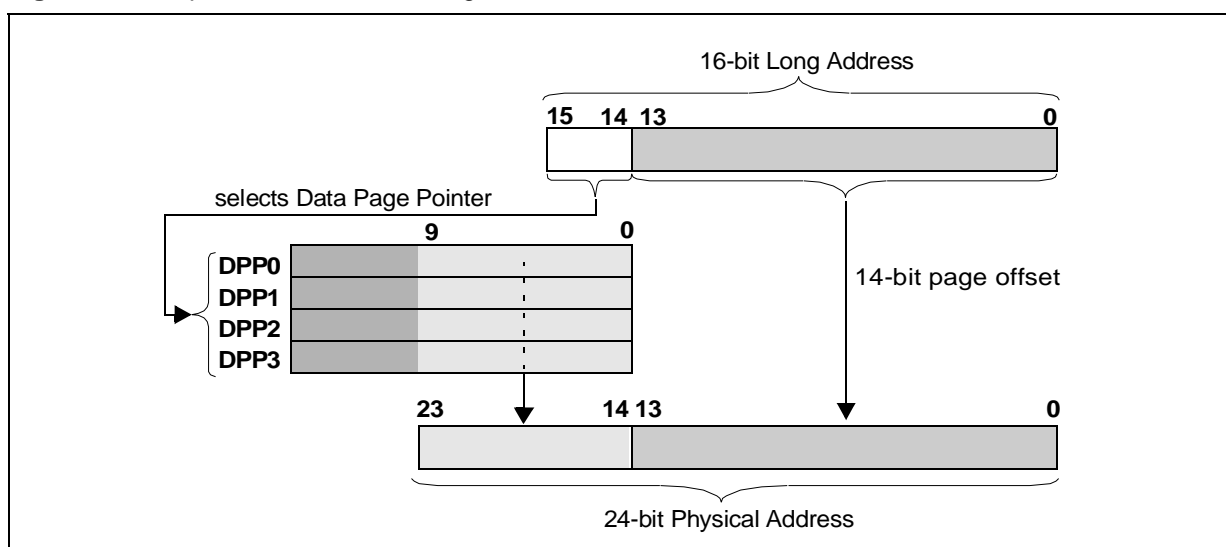
Long addressing mode uses one of the four DPP registers to specify a physical 18-bit or 24-bit address. Any word or byte data within the entire address space can be accessed in this mode. All devices support an override mechanism for the DPP addressing scheme (see section 2.1.3 - DPP override mechanism).

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 24-bit address (see Figure 1).

All ST10 devices support an address space of up to 16MByte, so only the lower ten bits of the selected DPP register content are concatenated with the 14-bit data page offset to build the physical address.

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.

Figure 1 : Interpretation of a 16-bit long address



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

Table 2 : Summary of long address modes

Mnemo	Physical Address	Long Address Range	Scope of Access
mem	(DPP0) mem^3FFFh	0000h...3FFFh	Any Word or Byte
	(DPP1) mem^3FFFh	4000h...7FFFh	
	(DPP2) mem^3FFFh	8000h...BFFFh	
	(DPP3) mem^3FFFh	C000h...FFFFh	
mem	pag mem^3FFFh	0000h...FFFFh (14-bit)	Any Word or Byte
mem	seg mem	0000h...FFFFh (16-bit)	Any Word or Byte

2.1.3 - DPP override mechanism

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) (see Figure 2).

2.1.4 - 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:

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).

$$\text{GPRAddress} = (\text{CP}) + 2 \times \text{ShortAddress}$$

2. Pre-decremented indirect address pointers ('-Rw') are decremented by a data-type-dependent value ($\Delta = 1$ for byte operations, $\Delta = 2$ for word operations), before the long 16-bit address is generated:

$$(\text{GPRAddress}) = (\text{GPRAddress}) - \Delta \text{ [optional step!]}$$

3. Calculate the long 16-bit (Rw + #data16) address by adding a constant value (if selected) to the content of the indirect address pointer:

$$\text{Long Address} = (\text{GPR Address}) + \text{Constant}$$

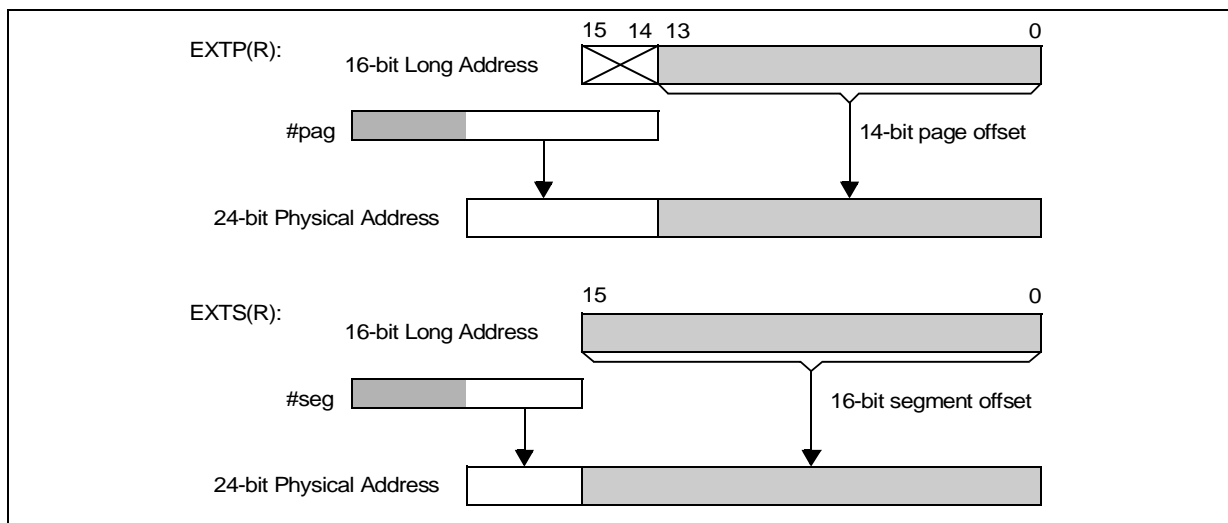
4. Calculate the physical 18-bit or 24-bit address using the resulting long address and the corresponding DPP register content (see long 'mem' addressing modes).

$$\text{Physical Address} = (\text{DPPi}) + \text{Long Address} \wedge 3FFFh$$

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

$$(\text{GPR Address}) = (\text{GPR Address}) + \Delta \text{ [optional step!]}$$

Figure 2 : Overriding the DPP mechanism



The following indirect addressing modes are provided:

Table 3 : 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 incremented by 2 or 1 (for word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically decremented by 2 or 1 (for word or byte data operations) before the access.
[Rw+#data ₁₆]	A 16-bit constant and the contents of the indirect address pointer are added before the long 16-bit address is calculated.

2.1.5 - Constants

The ST10 Family instruction set supports the use of wordwide or byte-wide 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 can be truncated to match the data format required for the operation:

ated to match the data format required for the operation:

Table 4 : Table of constants

Mnemonic	Word operation	Byte operation
#data ₃	0000 _h + data ₃	00 _h + data ₃
#data ₄	0000 _h + data ₄	00 _h + data ₄
#data ₈	0000 _h + data ₈	data ₈
#data ₁₆	data ₁₆	data ₁₆ ^ FF _h
#mask	0000 _h + mask	mask

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

2.1.6 - 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 5 : Branch target address summary

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000h...FFFEh
rel	(IP) = (IP) + 2*rel	-	rel = 00h...7Fh
	(IP) = (IP) + 2*(~rel+1)	-	rel = 80h...FFh
[Rw]	(IP) = ((CP) + 2*Rw)	-	Rw = 0...15
seg	-	(CSP) = seg	seg = 0...255
#trap ₇	(IP) = 0000h + 4*trap ₇	(CSP) = 0000h	trap ₇ = 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 support 256 different code segments, so only the eight 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".

2.2 - 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.

2.2.1 - Definition of measurement units

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

[f_{CPU}]: CPU operating frequency (may vary from 1MHz to 80MHz).

[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.

$$1 \text{ [State]} = 1/f_{CPU}[s] \quad ; \text{ for } f_{CPU} = \text{variable}$$

$$= 50[ns] \quad ; \text{ for } f_{CPU} = 20\text{MHz}$$

[ACT]: 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:

$$1 \cdot \text{ACT} = (2 + (15 - \text{MCTC}) + (1 - \text{MTTC})) \cdot \text{States}$$

$$= 100 \text{ n... } 900 \text{ ns ; for } f_{CPU} = 20\text{MHz}$$

For multiplexed external bus modes:

$$1 \cdot \text{ACT} = (3 + (15 - \text{MCTC}) + (1 - \text{MTTC})) \cdot \text{States}$$

$$= 150\text{ns ... } 950\text{ns ; for } f_{CPU} = 20\text{MHz}$$

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

$$T_{tot} = T_{I1} + T_{I2} + \dots + T_{In} + 6 \cdot \text{States}$$

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:

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

2.2.2 - 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.

Table 6 : Minimum instruction state times [Unit = ns]

Instruction	$T_{Imin} (ROM)$ [States]	$T_{Imin} (ROM)$ (20MHz CPU clk)
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 + #data ₁₆]	4	200
RET, RETI, RETP, RETS	4	200
TRAP	4	200
All other instructions	2	100

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:

- 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.

2.2.3 - 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 $1 * \text{State}$

Reading a GPR or any other directly addressed operand within the internal RAM space does NOT cause additional state time. 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:

```
In           : MOV R1, [R0+]           ; auto-increment R0
In+1       : MOV [R3], [R2]         ; if R2 points into the internal RAM space:
                                     ; TIadd = 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: $T_{Iadd} = 0, 1 * \text{State}$ or $2 * \text{States}$

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

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

```
In           : MOV T0, #1000h        ; write to Timer 0
In+1       : ADD R3, T1             ; read from Timer 1: TIadd = 1 * State
```

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

```
In           : ADD R0, #1000h        ; implicit modification of PSW flags
In+1       : BAND C, Z             ; read from PSW: TIadd = 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:

```
In           : MOV SP, #0FB00h      ; explicit update of the stack pointer
In+1       : SCXT R1, #1000h        ; implicit decrement of the stack pointer:
                                     ; TIadd = 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: $T_{Iadd} = 1 * \text{ACT}$

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: $T_{Iadd} = 0 * \text{State} \dots 1 * \text{ACT}$

Writing an external operand via a 16-bit wide data bus takes one additional ALE Cycle Time. For timing calculation of the 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)
....     : ....
In+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)
In+1    : ....                ; any non-aligned double word instruction
                                     ; (e.g. at location 12FEh)
In+2    : 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:

```
In      : BSET USR0           ; implicit modification of PSW flags
In+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.

2.3 - Instruction set summary

The following table lists the instruction mnemonic by hex-code with operand.

Table 7 : Instruction mnemonic by hex-code with operand

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
High																
Low	0x	ADDB	ADD	ADDB	ADD	ADDB	ADD	ADDB	ADD	ADDB	BFLDL BITOFF MASK, #data ₃	MUL Rw _n , Rw _m	ROL Rw _n , Rw _m			
	1x	ADDCB	ADDC	ADDCB	ADDC	ADDCB	ADDC	ADDCB	ADDC	ADDCB	BFLDH	MULLU Rw _n , #d ₄	ROL Rw _n , #d ₄			
	2x	SUBB	SUB	SUBB	SUB	SUBB	SUB	SUBB	SUB	SUBB	BCMP	PRIOR Rw _n , Rw _m	ROR Rw _n , Rw _m			
	3x	Rw _n , Rw _m	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	BITadd, BITadd					
	4x	SUBCB	SUBC	SUBCB	SUBC	SUBCB	SUBC	SUBCB	SUBC	SUBCB	BMOV/N		ROR Rw _n , #d ₄			
	5x	CMPB	CMP	CMPB	CMP	CMPB	CMP	CMPB	CMP	CMPB	BMOV	DIV Rw _n	SHL Rw _n , Rw _m			
	6x	Rw _n , Rw _m	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	BITadd, BITadd	DIVU Rw _n	SHL Rw _n , #d ₄			
	7x	XORB	XOR	XORB	XOR	XORB	XOR	XORB	XOR	XORB	BITadd, BITadd		SHR Rw _n , #d ₄			
	8x	ANDB	AND	ANDB	AND	ANDB	AND	ANDB	AND	ANDB	BAND		SHR Rw _n , Rw _m			
	9x	Rw _n , Rw _m	REG, MEM	REG, MEM	REG, MEM	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	REG, #data ₁₆	BITadd, BITadd		SHR Rw _n , #d ₄			
	Ax	ORB	OR	ORB	OR	ORB	OR	ORB	OR	ORB	JB					
	Bx	CoXXX	CMPD1	CoXXX	CMPD1	CoXXX	CMPD1	CoXXX	CMPD1	CoXXX	BITadd, REL					
	Cx	Rw _n	Rw _n , MEM	Rw _n	Rw _n , MEM	Rw _n	Rw _n , MEM	Rw _n	Rw _n , MEM	Rw _n	JNB	TRAP	JMPI			
	Dx	CPLB	CMPD2	CPLB	CMPD2	CPLB	CMPD2	CPLB	CMPD2	CPLB	JBC	#trap	ASHR Rw _n , Rw _m			
	Ex	NEGB	CMPD1	NEGB	CMPD1	NEGB	CMPD1	NEGB	CMPD1	NEGB	BITadd, BITadd	CALLI	ASHR Rw _n , #d ₄			
	Fx	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	Rw _n , #d ₄	JNBS	CALLR	ASHR Rw _n , #d ₄			
		CPLB	CMPD2	CPLB	CMPD2	CPLB	CMPD2	CPLB	CMPD2	CPLB	CALLA	RET	NOP			
		MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	CC, CADDR	RETS	EXTP(R)/ Rw _n , #d ₂			
		REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	CALLS	RETS	EXTP(R)/ Rw _n , #d ₂			
		MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	MOVBS	SEG, CADDR	RETS	EXTP(R)/ Rw _n , #d ₂			
		PCALL	PCALL	PCALL	PCALL	PCALL	PCALL	PCALL	PCALL	PCALL	JMPA	RETP	PUSH			
		REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	REG, CADDR	CC, CADDR	REG	REG			
		MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	JMPS	REG	REG			
		REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	REG, MEM	SEG, CADDR	RETI	POP			
		MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	SEG, CADDR	RETI	POP			
		Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄						
		MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV						
		Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄	Rw _n , #data ₄						



ST10 FAMILY PROGRAMMING MANUAL

Table 8 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 8 : Mnemonic vs address mode & number of bytes

Mnemonic	Addressing Modes	Bytes	Mnemonic	Addressing Modes	Bytes
ADD[B]	Rw_n^1, Rw_m^1	2	CPL[B]	Rw_n^1	2
ADDC[B]	$Rw_n^1, [Rw_i]$	2	NEG[B]		
AND[B]	$Rw_n^1, [Rw_i+]$	2	DIV	Rw_n	2
OR[B]	$Rw_n^1, \#data_3$	2	DIVL		
SUB[B]	reg, $\#data_{16}$	4	DIVLU		
SUBC[B]	reg, mem	4	DIVU		
XOR[B]	mem, reg	4	MUL	Rw_n, Rw_m	2
ASHR	Rw_n, Rw_m	2	MULU		
ROL / ROR	$Rw_n, \#data_4$	2	CMPD1/2	$Rw_n, \#data_4$	2
SHL / SHR			CMPI1/2	$Rw_n, \#data_{16}$	4
BAND	bitaddr _{Z,z} , bitaddr _{Q,q}	4		Rw_n, mem	4
BCMP			CMP[B]	Rw_n, Rw_m^1	2
BMOV				$Rw_n, [Rw_i]^1$	2
BMOVN				$Rw_n, [Rw_i+]$ ¹	2
BOR / BXOR				$Rw_n, \#data_3^1$	2
BCLR	bitaddr _{Q,q}	2		reg, $\#data_{16}$	4
BSET				reg, mem	4
BFLDH	bitoff _Q , $\#mask_8, \#data_8$	4	CALLA	cc, caddr	4
BFLDL			JMPA		
MOV[B]	Rw_n^1, Rw_m^1	2	CALLI	cc, [Rw_n]	2
	$Rw_n^1, \#data_4$	2	JMPI		
	$Rw_n^1, [Rw_m]$	2	CALLS	seg, caddr	4
	$Rw_n^1, [Rw_m+]$	2	JMPS		
	$[Rw_m], Rw_n^1$	2	CALLR	rel	2
	$[-Rw_m], Rw_n^1$	2	JMPR	cc, rel	2
	$[Rw_n], [Rw_m]$	2	JB	bitaddr _{Q,q} , rel	4
	$[Rw_n+], [Rw_m]$	2	JBC		
	reg, $\#data_{16}$	4	JNB		
	$Rw_n, [Rw_m+\#data_{16}]^1$	4	JNBS		
	$[Rw_m+\#data_{16}], Rw_n^1$	4	PCALL	reg, caddr	4
	$[Rw_n], mem$	4	POP	reg	2
	mem, [Rw_n]	4	PUSH		
	reg, mem	4	RETP		
	mem, reg	4	SCXT	reg, $\#data_{16}$	4
				reg, mem	4
			PRIOR	Rw_n, Rw_m	2

Table 8 : Mnemonic vs address mode & number of bytes (continued)

Mnemonic	Addressing Modes	Bytes	Mnemonic	Addressing Modes	Bytes
MOVBS	Rw _n , Rb _m	2	TRAP	#trap7	2
MOVBS	reg, mem	4	ATOMIC	#data ₂	2
MOVBS	mem, reg	4	EXTR		
EXTS	Rw _m , #data ₂	2	EXTP	Rw _m , #data ₂	2
EXTSR	#seg, #data ₂	4	EXTPR	#pag, #data ₂	4
NOP	-	2	SRST/IDLE	-	4
RET			PWRDN		
RETI			SRVWDT		
RETS	=		DISWDT		
			EINIT		

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

2.4 - Instruction set ordered by functional group

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_x: Immediate constant (the number of significant bits that can be user-specified is given by the 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 8-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:

#pag: Immediate 10-bit page address.

#seg: Immediate 8-bit segment address.

Branch condition codes

cc: Symbolically specifiable condition codes

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

Table 9 : Arithmetic instructions

Mnemonic	Description	Int.ROM	Int.RAM	16-bit N-Mux	16-bit Mux	8-bit N-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, #data ₃	Add immediate word data to direct GPR	2	6	2	3	4	6	2
ADD reg, #data ₁₆	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, #data ₃	Add immediate byte data to direct GPR	2	6	2	3	4	6	2
ADDB reg, #data ₁₆	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, #data ₃	Add immediate word data to direct GPR with Carry	2	6	2	3	4	6	2
ADDC reg, #data ₁₆	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

Table 9 : Arithmetic instructions (continued)

Mnemonic	Description	Int.ROM	Int.RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
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
ADDCB Rb, #data ₃	Add immediate byte data to direct GPR with Carry	2	6	2	3	4	6	2
ADDCB reg, #data ₁₆	Add immediate byte data to direct register with Carry	2	8	4	6	8	12	4
ADDCB reg, mem	Add direct byte memory to direct register with Carry	2	8	4	6	8	12	4
ADDCB mem, 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
DIVLU Rw	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, #data ₃	Subtract immediate word data from direct GPR	2	6	2	3	4	6	2
SUB reg, #data ₁₆	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
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, #data ₃	Subtract immediate byte data from direct GPR	2	6	2	3	4	6	2
SUBB reg, #data ₁₆	Subtract immediate byte data from direct register	2	8	4	6	8	12	4

Table 9 : Arithmetic instructions (continued)

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

Table 10 : Logical instructions

Mnemonic		Description	Int.ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-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, #data ₃	Bitwise AND immediate word data with direct GPR	2	6	2	3	4	6	2
AND	reg, #data ₁₆	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

Table 10 : Logical instructions (continued)

Mnemonic		Description	Int ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit MUX	Bytes
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, #data ₃	Bitwise AND immediate byte data with direct GPR	2	6	2	3	4	6	2
ANDB	reg, #data ₁₆	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
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, #data ₃	Bitwise OR immediate word data with direct GPR	2	6	2	3	4	6	2
OR	reg, #data ₁₆	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 and post-increment source pointer by 1	2	6	2	3	4	6	2
ORB	Rb, #data ₃	Bitwise OR immediate byte data with direct GPR	2	6	2	3	4	6	2
ORB	reg, #data ₁₆	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, #data ₃	Bitwise XOR immediate word data with direct GPR	2	6	2	3	4	6	2
XOR	reg, #data ₁₆	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, #data ₃	Bitwise XOR immediate byte data with direct GPR	2	6	2	3	4	6	2
XORB	reg, #data ₁₆	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 11 : Boolean bit map instructions (continued)

Mnemonic	Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-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, #mask ₈ , #data ₈	Bitwise modify masked high byte of bit-addressable direct word memory with immediate data	2	8	4	6	8	12	4
BFLDL bitoff, #mask ₈ , #data ₈	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
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, #data ₃	Compare immediate word data to direct GPR	2	6	2	3	4	6	2
CMP reg, #data ₁₆	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
CMPB Rb, Rb	Compare direct byte GPR to direct GPR	2	6	2	3	4	6	2
CMPB Rb, [Rw]	Compare indirect byte memory to direct GPR	2	6	2	3	4	6	2
CMPB Rb, [Rw+]	Compare indirect byte memory to direct GPR and post-increment source pointer by 1	2	6	2	3	4	6	2
CMPB Rb, #data ₃	Compare immediate byte data to direct GPR	2	6	2	3	4	6	2
CMPB reg, #data ₁₆	Compare immediate byte data to direct register	2	8	4	6	8	12	4
CMPB reg, mem	Compare direct byte memory to direct register	2	8	4	6	8	12	4

Table 12 : Compare and loop instructions (continued)

Mnemonic		Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
CMPD1	Rw, #data ₄	Compare immediate word data to direct GPR and decrement GPR by 1	2	6	2	3	4	6	2
CMPD1	Rw, #data ₁₆	Compare immediate word data to direct GPR and decrement GPR by 1	2	8	4	6	8	12	4
CMPD1	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 1	2	8	4	6	8	12	4
CMPD2	Rw, #data ₄	Compare immediate word data to direct GPR and decrement GPR by 2	2	6	2	3	4	6	2
CMPD2	Rw, #data ₁₆	Compare immediate word data to direct GPR and decrement GPR by 2	2	8	4	6	8	12	4
CMPD2	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 2	2	8	4	6	8	12	4
CMP11	Rw, #data ₄	Compare immediate word data to direct GPR and increment GPR by 1	2	6	2	3	4	6	2
CMP11	Rw, #data ₁₆	Compare immediate word data to direct GPR and increment GPR by 1	2	8	4	6	8	12	4
CMP11	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 1	2	8	4	6	8	12	4
CMP12	Rw, #data ₄	Compare immediate word data to direct GPR and increment GPR by 2	2	6	2	3	4	6	2
CMP12	Rw, #data ₁₆	Compare immediate word data to direct GPR and increment GPR by 2	2	8	4	6	8	12	4
CMP12	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 2	2	8	4	6	8	12	4

Table 13 : Prioritize instructions

Mnemonic		Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
PRIOR	Rw, 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 14 : Shift and rotate instructions (continued)

Mnemonic		Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-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, #data ₄	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, #data ₄	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, #data ₄	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, #data ₄	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, #data ₄	Shift right direct word GPR; number of shift cycles specified by immediate data	2	6	2	3	4	6	2

Table 15 : Data movement instructions

Mnemonic		Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
MOV	Rw, Rw	Move direct word GPR to direct GPR	2	6	2	3	4	6	2
MOV	Rw, #data ₄	Move immediate word data to direct GPR	2	6	2	3	4	6	2
MOV	reg, #data ₁₆	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 memory	2	6	2	3	4	6	2
MOV	[Rw+], [Rw]	Move indirect word memory to indirect memory & post-increment destination pointer by 2	2	6	2	3	4	6	2
MOV	[Rw], [Rw+]	Move indirect word memory to indirect memory & post-increment source pointer by 2	2	6	2	3	4	6	2

Table 15 : Data movement instructions (continued)

Mnemonic		Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
MOV	Rw, [Rw+ #data ₁₆]	Move indirect word memory by base plus constant to direct GPR	4	10	6	8	10	14	4
MOV	[Rw+ #data ₁₆], 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, #data ₄	Move immediate byte data to direct GPR	2	6	2	3	4	6	2
MOVB	reg, #data ₁₆	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+ #data ₁₆]	Move indirect byte memory by base plus constant to direct GPR	4	10	6	8	10	14	4
MOVB	[Rw+ #data ₁₆], 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
MOVBS	mem, reg	Move direct byte register with sign extension to direct word memory	2	8	4	6	8	12	4
MOVBS	Rw, Rb	Move direct byte GPR with zero extension to direct word GPR	2	6	2	3	4	6	2
MOVBS	reg, mem	Move direct byte memory with zero extension to direct word register	2	8	4	6	8	12	4
MOVBS	mem, reg	Move direct byte register with zero extension to direct word memory	2	8	4	6	8	12	4

Table 16 : Jump and Call Instructions (continued)

Mnemonic	Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
CALLA cc, 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
CALLR rel	Call relative subroutine	4	8	4	5	6	8	2
CALLS seg, 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
PCALL reg, 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 17 : System Stack Instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit	16-bit	8-bit	8-bit	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, #data ₁₆	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 18 : Return Instructions

Mnemonic	Description	Int. ROM	Int. RAM	16-bit	16-bit	8-bit	8-bit	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 19 : System Control Instructions (continued)

Mnemonic	Description	Int. ROM	Int. RAM	16-bit N-Mux	16-bit Mux	8-bit N-Mux	8-bit Mux	Bytes
ATOMIC #data ₂	Begin ATOMIC sequence ¹	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 #data ₂	Begin EXTENDED Register sequence ¹	2	6	2	3	4	6	2
EXTP Rw, #data ₂	Begin EXTENDED Page sequence ¹	2	6	2	3	4	6	2
EXTP #pag, #data ₂	Begin EXTENDED Page sequence ¹	2	8	4	6	8	12	4
EXTPR Rw, #data ₂	Begin EXTENDED Page and Register sequence ¹	2	6	2	3	4	6	2
EXTPR #pag, #data ₂	Begin EXTENDED Page and Register sequence ¹	2	8	4	6	8	12	4
EXTS Rw, #data ₂	Begin EXTENDED Segment sequence ¹	2	6	2	3	4	6	2
EXTS #seg, #data ₂	Begin EXTENDED Segment sequence ¹	2	8	4	6	8	12	4
EXTSR Rw, #data ₂	Begin EXTENDED Segment and Register sequence ¹	2	6	2	3	4	6	2
EXTSR #seg, #data ₂	Begin EXTENDED Segment and Register sequence ¹	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 is 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

Note 1. The EXT instructions override the standard DPP addressing scheme.

Table 20 : Miscellaneous instructions

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

2.5 - 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. Some 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. Some instructions are encoded by means of additional bits in the operand field of the instruction.

00xx.xxxx: EXTS or ATOMIC

01xx.xxxx: EXTP

00xx.xxxx: EXTS or ATOMIC

10xx.xxxx: EXTSR or EXTR

11xx.xxxx: EXTPR

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 "bitaddr_{Q,q}" (where q=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 21 : Instruction set ordered by Hex code

Hex- code	Number of Bytes	Mnemonic	Operand
00	2	ADD	Rw _n , Rw _m
01	2	ADDB	Rb _n , Rb _m
02	4	ADD	reg, mem
03	4	ADDB	reg, mem
04	4	ADD	mem, reg
05	4	ADDB	mem, reg
06	4	ADD	reg, #data ₁₆
07	4	ADDB	reg, #data ₁₆
08	2	ADD	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
09	2	ADDB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
0A	4	BFLDL	bitoff _Q , #mask ₈ , #data ₈
0B	2	MUL	Rw _n , Rw _m
0C	2	ROL	Rw _n , Rw _m
0D	2	JMPR	cc_UC, rel
0E	2	BCLR	bitaddr _{Q,0}
0F	2	BSET	bitaddr _{Q,0}
10	2	ADDC	Rw _n , Rw _m
11	2	ADDCB	Rb _n , Rb _m

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
12	4	ADDC	reg, mem
13	4	ADDCB	reg, mem
14	4	ADDC	mem, reg
15	4	ADDCB	mem, reg
16	4	ADDC	reg, #data ₁₆
17	4	ADDCB	reg, #data ₁₆
18	2	ADDC	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
19	2	ADDCB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
1A	4	BFLDH	bitoff _Q , #mask ₈ , #data ₈
1B	2	MULU	Rw _n , Rw _m
1C	2	ROL	Rw _n , #data ₄
1D	2	JMPR	cc_NET, rel
1E	2	BCLR	bitaddr _{Q.1}
1F	2	BSET	bitaddr _{Q.1}
20	2	SUB	Rw _n , Rw _m
21	2	SUBB	Rb _n , Rb _m
22	4	SUB	reg, mem
23	4	SUBB	reg, mem
24	4	SUB	mem, reg
25	4	SUBB	mem, reg
26	4	SUB	reg, #data ₁₆
27	4	SUBB	reg, #data ₁₆
28	2	SUB	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
29	2	SUBB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
2A	4	BCMP	bitaddr _{Z.Z} , bitaddr _{Q.q}
2B	2	PRIOR	Rw _n , Rw _m
2C	2	ROR	Rw _n , Rw _m
2D	2	JMPR	cc_EQ, rel or cc_Z, rel
2E	2	BCLR	bitaddr _{Q.2}
2F	2	BSET	bitaddr _{Q.2}
30	2	SUBC	Rw _n , Rw _m
31	2	SUBCB	Rb _n , Rb _m
32	4	SUBC	reg, mem
33	4	SUBCB	reg, mem

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
34	4	SUBC	mem, reg
35	4	SUBCB	mem, reg
36	4	SUBC	reg, #data ₁₆
37	4	SUBCB	reg, #data ₁₆
38	2	SUBC	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
39	2	SUBCB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
3A	4	BMOVN	bitaddr _{Z,Z} , bitaddr _{Q,q}
3B	-	-	-
3C	2	ROR	Rw _n , #data ₄
3D	2	JMPR	cc_NE, rel or cc_NZ, rel
3E	2	BCLR	bitaddr _{Q,3}
3F	2	BSET	bitaddr _{Q,3}
40	2	CMP	Rw _n , Rw _m
41	2	CMPB	Rb _n , Rb _m
42	4	CMP	reg, mem
43	4	CMPB	reg, mem
44	-	-	-
45	-	-	-
46	4	CMP	reg, #data ₁₆
47	4	CMPB	reg, #data ₁₆
48	2	CMP	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
49	2	CMPB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
4A	4	BMOV	bitaddr _{Z,Z} , bitaddr _{Q,q}
4B	2	DIV	Rw _n
4C	2	SHL	Rw _n , Rw _m
4D	2	JMPR	cc_V, rel
4E	2	BCLR	bitaddr _{Q,4}
4F	2	BSET	bitaddr _{Q,4}
50	2	XOR	Rw _n , Rw _m
51	2	XORB	Rb _n , Rb _m
52	4	XOR	reg, mem
53	4	XORB	reg, mem
54	4	XOR	mem, reg
55	4	XORB	mem, reg

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
56	4	XOR	reg, #data ₁₆
57	4	XORB	reg, #data ₁₆
58	2	XOR	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
59	2	XORB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
5A	4	BOR	bitaddr _{Z,z} , bitaddr _{Q,q}
5B	2	DIVU	Rw _n
5C	2	SHL	Rw _n , #data ₄
5D	2	JMPR	cc_NV, rel
5E	2	BCLR	bitaddr _{Q,5}
5F	2	BSET	bitaddr _{Q,5}
60	2	AND	Rw _n , Rw _m
61	2	ANDB	Rb _n , Rb _m
62	4	AND	reg, mem
63	4	ANDB	reg, mem
64	4	AND	mem, reg
65	4	ANDB	mem, reg
66	4	AND	reg, #data ₁₆
67	4	ANDB	reg, #data ₁₆
68	2	AND	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
69	2	ANDB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
6A	4	BAND	bitaddr _{Z,z} , bitaddr _{Q,q}
6B	2	DIVL	Rw _n
6C	2	SHR	Rw _n , Rw _m
6D	2	JMPR	cc_N, rel
6E	2	BCLR	bitaddr _{Q,6}
6F	2	BSET	bitaddr _{Q,6}
70	2	OR	Rw _n , Rw _m
71	2	ORB	Rb _n , Rb _m
72	4	OR	reg, mem
73	4	ORB	reg, mem
74	4	OR	mem, reg
75	4	ORB	mem, reg
76	4	OR	reg, #data ₁₆
77	4	ORB	reg, #data ₁₆

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
78	2	OR	Rw _n , [Rw _i +] or Rw _n , [Rw _i] or Rw _n , #data ₃
79	2	ORB	Rb _n , [Rw _i +] or Rb _n , [Rw _i] or Rb _n , #data ₃
7A	4	BXOR	bitaddr _{Z,z} , bitaddr _{Q,q}
7B	2	DIVLU	Rw _n
7C	2	SHR	Rw _n , #data ₄
7D	2	JMPR	cc_NN, rel
7E	2	BCLR	bitaddr _{Q,7}
7F	2	BSET	bitaddr _{Q,7}
80	2	CMPI1	Rw _n , #data ₄
81	2	NEG	Rw _n
82	4	CMPI1	Rw _n , mem
83	4	CoXXX ¹	Rw _n , [Rw _m ⊗]
84	4	MOV	[Rw _n], mem
85	-	-	-
86	4	CMPI1	Rw _n , #data ₁₆
87	4	IDLE	
88	2	MOV	[-Rw _m], Rw _n
89	2	MOVB	[-Rw _m], Rb _n
8A	4	JB	bitaddr _{Q,q} , rel
8B	-	-	-
8C	-	-	-
8D	2	JMPR	cc_C, rel or cc_ULT, rel
8E	2	BCLR	bitaddr _{Q,8}
8F	2	BSET	bitaddr _{Q,8}
90	2	CMPI2	Rw _n , #data ₄
91	2	CPL	Rw _n
92	4	CMPI2	Rw _n , mem
93	4	CoXXX ¹	[IDX _i ⊗], [Rw _n ⊗]
94	4	MOV	mem, [Rw _n]
95	-	-	-
96	4	CMPI2	Rw _n , #data ₁₆
97	4	PWRDN	
98	2	MOV	Rw _n , [Rw _m +]
99	2	MOVB	Rb _n , [Rw _m +]

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
9A	4	JNB	bitaddr _{Q,q} , rel
9B	2	TRAP	#trap7
9C	2	JMPI	cc, [Rw _n]
9D	2	JMPR	cc_NC, rel or cc_UGE, rel
9E	2	BCLR	bitaddr _{Q,9}
9F	2	BSET	bitaddr _{Q,9}
A0	2	CMPD1	Rw _n , #data ₄
A1	2	NEGB	Rb _n
A2	4	CMPD1	Rw _n , mem
A3	4	CoXXX ¹	Rw _n , Rw _m
A4	4	MOVB	[Rw _n], mem
A5	4	DISWDT	
A6	4	CMPD1	Rw _n , #data ₁₆
A7	4	SRVWDT	
A8	2	MOV	Rw _n , [Rw _m]
A9	2	MOVB	Rb _n , [Rw _m]
AA	4	JBC	bitaddr _{Q,q} , rel
AB	2	CALLI	cc, [Rw _n]
AC	2	ASHR	Rw _n , Rw _m
AD	2	JMPR	cc_SGT, rel
AE	2	BCLR	bitaddr _{Q,10}
AF	2	BSET	bitaddr _{Q,10}
B0	2	CMPD2	Rw _n , #data ₄
B1	2	CPLB	Rb _n
B2	4	CMPD2	Rw _n , mem
B3	4	CoSTORE ¹	[Rw _n ⊗], CoReg
B4	4	MOVB	mem, [Rw _n]
B5	4	EINIT	
B6	4	CMPD2	Rw _n , #data ₁₆
B7	4	SRST	
B8	2	MOV	[Rw _m], Rw _n
B9	2	MOVB	[Rw _m], Rb _n
BA	4	JNBS	bitaddr _{Q,q} , rel
BB	2	CALLR	rel

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
BC	2	ASHR	Rw _n , #data ₄
BD	2	JMPR	cc_SLE, rel
BE	2	BCLR	bitaddr _{Q.11}
BF	2	BSET	bitaddr _{Q.11}
C0	2	MOVBZ	Rb _n , Rb _m
C1	-	-	-
C2	4	MOVBZ	reg, mem
C3	4	CoSTORE ¹	Rw _n , CoReg
C4	4	MOV	[Rw _m +#data ₁₆], Rw _n
C5	4	MOVBZ	mem, reg
C6	4	SCXT	reg, #data ₁₆
C7	-	-	-
C8	2	MOV	[Rw _n], [Rw _m]
C9	2	MOVB	[Rw _n], [Rw _m]
CA	4	CALLA	cc, caddr
CB	2	RET	
CC	2	NOP	
CD	2	JMPR	cc_SLT, rel
CE	2	BCLR	bitaddr _{Q.12}
CF	2	BSET	bitaddr _{Q.12}
D0	2	MOVBS	Rb _n , Rb _m
D1	2	ATOMIC/EXTR	#data ₂
D2	4	MOVBS	reg, mem
D3	4	CoMOV ¹	[IDX _i ⊗], [Rw _n ⊗]
D4	4	MOV	Rw _n , [Rw _m +#data ₁₆]
D5	4	MOVBS	mem, reg
D6	4	SCXT	reg, mem
D7	4	EXTP(R)/EXTS(R)	#pag, #data ₂
D8	2	MOV	[Rw _n +], [Rw _m]
D9	2	MOVB	[Rw _n +], [Rw _m]
DA	4	CALLS	seg, caddr
DB	2	RETS	
DC	2	EXTP(R)/EXTS(R)	Rw _m , #data ₂
DD	2	JMPR	cc_SGE, rel

Table 21 : Instruction set ordered by Hex code (continued)

Hex- code	Number of Bytes	Mnemonic	Operand
DE	2	BCLR	bitaddr _{Q.13}
DF	2	BSET	bitaddr _{Q.13}
E0	2	MOV	Rw _n , #data ₄
E1	2	MOVB	Rb _n , #data ₄
E2	4	PCALL	reg, caddr
E3	-	-	-
E4	4	MOVB	[Rw _m +#data ₁₆], Rb _n
E5	-	-	-
E6	4	MOV	reg, #data ₁₆
E7	4	MOVB	reg, #data ₁₆
E8	2	MOV	[Rw _n], [Rw _m +]
E9	2	MOVB	[Rw _n], [Rw _m +]
EA	4	JMPA	cc, caddr
EB	2	RETP	reg
EC	2	PUSH	reg
ED	2	JMPR	cc_UGT, rel
EE	2	BCLR	bitaddr _{Q.14}
EF	2	BSET	bitaddr _{Q.14}
F0	2	MOV	Rw _n , Rw _m
F1	2	MOVB	Rb _n , Rb _m
F2	4	MOV	reg, mem
F3	4	MOVB	reg, mem
F4	4	MOVB	Rb _n , [Rw _m +#data ₁₆]
F5	-	-	-
F6	4	MOV	mem, reg
F7	4	MOVB	mem, reg
F8	-	-	-
F9	-	-	-
FA	4	JMPS	seg, caddr
FB	2	RETI	
FC	2	POP	reg
FD	2	JMPR	cc_ULE, rel
FE	2	BCLR	bitaddr _{Q.15}
FF	2	BSET	bitaddr _{Q.15}

Note 1. This instruction only applies to products including the MAC.

2.6 - 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:

2.6.1 - Instruction name

Specifies the mnemonic opcode of the instruction.

2.6.2 - 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 operand syntax depends on the addressing mode. All of the available addressing modes are

summarized at the end of each single instruction description.

2.6.3 - Operation

The following symbols are used to represent data movement, arithmetic or logical operators (see Table 22).

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).

Table 22 : Instruction operation symbols

			operator (opY)
Diadic operations	(opx) <-- (opy)	(opY)	is MOVED into (opX)
	(opx) + (opy)	(opX)	is ADDED to (opY)
	(opx) - (opy)	(opY)	is SUBTRACTED from (opX)
	(opx) * (opy)	(opX)	is MULTIPLIED by (opY)
	(opx) / (opy)	(opX)	is DIVIDED by (opY)
	(opx) ^ (opy)	(opX)	is logically ANDed with (opY)
	(opx) v (opy)	(opX)	is logically ORed with (opY)
	(opx) ⊕ (opy)	(opX)	is logically EXCLUSIVELY ORed with (opY)
	(opx) <--> (opy)	(opX)	is COMPARED against (opY)
(opx) mod (opy)	(opX)	is divided MODULO (opY)	
Monadic operations			operator (opX)
	(opx) ¬	(opX)	is logically COMPLEMENTED

The following abbreviations are used to describe operands:

Table 23 : Operand abbreviations

Abbreviation	Description
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 flag in the PSW register.
V	Overflow 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.

2.6.4 - 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.

2.6.5 - Description

Describes the operation of the instruction.

2.6.6 - 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 24 : 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 = 0	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 v C) = 1	Unsigned less than or equal	Fh
cc_UGE	C = 0	Unsigned greater than or equal	9h
cc_UGT	(Z v C) = 0	Unsigned greater than	Eh
cc_SLT	(N ⊕ V) = 1	Signed less than	Ch
cc_SLE	(Z v (N ⊕ V)) = 1	Signed less than or equal	Bh
cc_SGE	(N ⊕ V) = 0	Signed greater than or equal	Dh
cc_SGT	(Z v (N ⊕ V)) = 0	Signed greater than	Ah
cc_NET	(Z v E) = 0	Not equal AND not end of table	1h

2.6.7 - 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 (see Table 25).

If the PSW register is specified as the destination operand of an instruction, the flags can not be interpreted as described.

This is because the PSW register is modified according to the data format of the instruction:

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

In all cases, if the PSW is the destination operand of an instruction, the PSW flags do NOT represent the flags of this instruction, in the normal way.

Table 25 : List of flags

Symbol	Description
*	The flag is set according to the following standard rules
	N = 1 : Most significant bit of the result is set
	N = 0 : Most significant bit 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
"O"	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.
" \bar{B} "	The flag contains the complemented value of the specified bit operand

2.6.8 - 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 3* 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 26 : 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 (Rw _i)
ss	8-bit code segment number (seg).
..##	2-bit immediate constant (#data ₂)
:.###	3-bit immediate constant (#data ₃)
c	4-bit condition code specification (cc)
n	4-bit short GPR address (Rw _n or Rb _n)
m	4-bit short GPR address (Rw _m or Rb _m)
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 (#data ₄)
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 (#data ₈)
@ @	8-bit immediate constant (#mask ₈)
pp 0:00pp	10-bit page address (#pag ₁₀)
MM MM	16-bit address (mem or caddr; low byte, high byte)
## ##	16-bit immediate constant (#data ₁₆ ; 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 (see Figure 3).

2.7 - 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

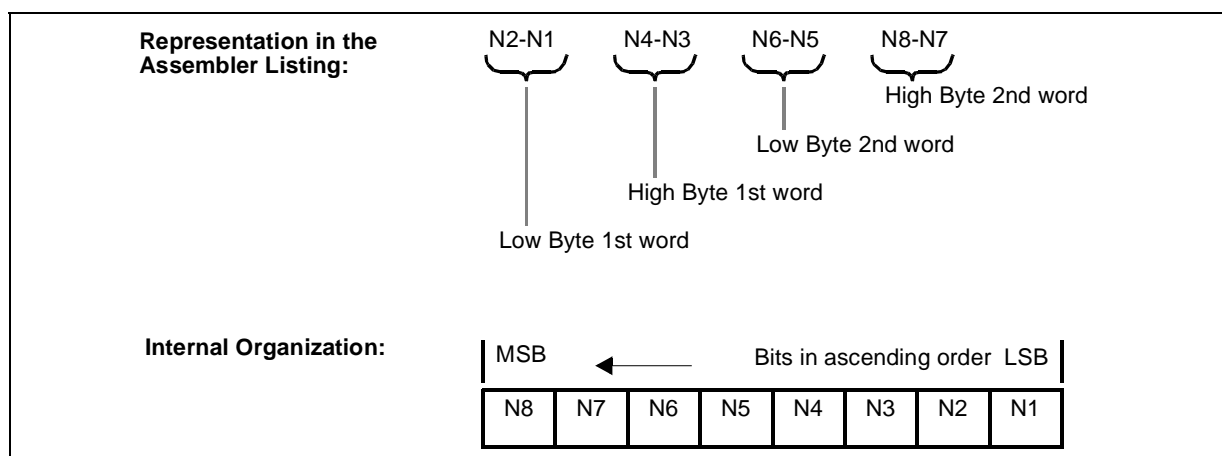
be used with the ATOMIC and EXTended instructions.

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: When using the ATOMIC and EXTended instructions with other system control or branch instructions.

CAUTION: 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.

Figure 3 : Instruction format representation



2.8 - Instruction descriptions

This section contains 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.

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.

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.

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

ADDCB	Integer Addition with Carry	
Syntax	ADDCB	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.

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)	<-- (op1) ^ (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.

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.

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

ATOMIC Begin ATOMIC Sequence

Syntax ATOMIC op1

Operation (count) <-- (op1) [1 ≤ op1 ≤ 4]
 Disable interrupts and Class A traps
 DO WHILE ((count) ≠ 0 AND Class_B_trap_condition ≠ TRUE)
 Next Instruction
 (count) <-- (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 2.7 - ATOMIC and EXTENDED instructions on page 38).

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

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.

Flags

E	Z	V	C	N
0	\bar{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,q}	qE QQ	2

BFLDH Bit Field High Byte

Syntax	BFLDH	op1, op2, op3
Operation	(tmp)	<-- (op1)
	(high byte (tmp))	<-- ((high byte (tmp) ^ ~op2) v op3)
	(op1)	<-- (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'.

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) <-- (op1)
 (low byte (tmp)) <-- ((low byte (tmp) ^ ¬op2) v op3)
 (op1) <-- (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'.

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)	<-- (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.

Flags

E	Z	V	C	N
0	\bar{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 & Negate**

Syntax BMOVN op1, op2

Operation (op1) <-- \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.

Flags

E	Z	V	C	N
0	\bar{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) <-- (op1) v (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.

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)	<-- 1
Data Types	BIT	

Description

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

Flags

E	Z	V	C	N
0	\bar{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	bitaddr _{Q,q}	qF QQ	2

BXOR	Bit Logical XOR	
Syntax	BXOR	op1, op2
Operation	(op1)	<-- (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.

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

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 24 - page 35.

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]	2

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 24 - page 35.

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 24 - page 35.

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	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.

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

Flag

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 & 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.

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 & Decrement by 2

Syntax	CMPD2	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 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.

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 & 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.

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 & 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.

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) <-- \neg (op1)
Data Types WORD

Description

Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

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)	$\leftarrow \neg(\text{op1})$
Data Types	BYTE	

Description

Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

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

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.

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).

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).

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).

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).

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 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.

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

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 (see Section 2.7 - ATOMIC and EXTENDED instructions on page 38).

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 & 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 2.7 - ATOMIC and EXTended instructions on page 38).

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

EXTR Begin **EXT**ended 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 2.7 - ATOMIC and EXTended instructions on page 38).

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

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 2.7 - ATOMIC and EXTended instructions on page 38).

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 & 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 2.7 - ATOMIC and EXTended instructions on page 38).

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.

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.

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	4

JBC Relative Jump if Bit Set & 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.

Flags

E	Z	V	C	N
0	\bar{B}	0	0	B

E Always cleared
 Z Contains 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
JBC bitaddr _{Q,q} , rel	AA QQ rr q0	4

JMPA Absolute Conditional Jump

Syntax JMPA op1, op2

Operation IF (op1) = 1 THEN
 (IP) <-- 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 JMPA instruction is executed normally.

Condition Codes

See Condition code Table 24 - page 35.

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

JMPI **Indirect Conditional Jump**

Syntax JMPI op1, op2

Operation IF (op1) = 1 THEN
(IP) <-- (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 24 - page 35.

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]	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 24 - page 35.

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.

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.

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 & 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.

Flags

E	Z	V	C	N
0	\bar{B}	0	0	B

E Always cleared.

Z Contains 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
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 flags are updated accordingly.

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 flags are updated accordingly.

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)	<-- (op2)
	IF (op2 ₇) = 1 THEN	(high byte op1) <-- FF _h
	ELSE	(high byte op1) <-- 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 flags are updated accordingly.

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	Rb _n , Rb _m	D0 mn	2
MOVBS	reg, mem	D2 RR MM MM	4
MOVBS	mem, reg	D5 RR MM MM	4

MUL	Signed Multiplication	
Syntax	MUL	op1, op2
Operation	(MD)	<-- (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.

Flags

E	Z	V	C	N
0	*	S	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	2

MULU **Unsigned Multiplication**

Syntax	MULU	op1, op2
Operation	(MD)	$\leftarrow (op1) * (op2)$
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.

Flags

E	Z	V	C	N
0	*	S	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	1B nm	2

NEG	Integer Two's Complement	
Syntax	NEG	op1
Operation	(op1)	$\leftarrow 0 - (\text{op1})$
Data Types	WORD	

Description

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

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	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.

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.

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)	<-- (op1) v (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.

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)	<-- (op1) v (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.

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 & 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.

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
PCALL	reg, caddr	E2 RR MM MM

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.

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	2

PRIOR Prioritize Register

Syntax PRIOR op1, op2

Operation

```

    (tmp) <-- (op2)
    (count) <-- 0
    DO WHILE (tmp15) ≠ 1 AND (count) ≠ 15 AND (op2) ≠ 0
        (tmpn) <-- (tmpn-1)
        (count) <-- (count) + 1
    END WHILE
    (op1) <-- (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 most significant bit 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.

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

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.

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 ($\overline{\text{NMI}}$) is in the low state. Otherwise, this instruction has no effect.

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.

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.

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

ST10 FAMILY PROGRAMMING MANUAL

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.

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) <-- (op2)
 (C) <-- 0
 DO WHILE (count) ≠ 0
 (C) <-- (op1₁₅)
 (op1_n) <-- (op1_{n-1}) [n=1...15]
 (op1₀) <-- (C)
 (count) <-- (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.

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 most significant bit 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) <-- (op2)
(C) <-- 0
(V) <-- 0
DO WHILE (count) ≠ 0
    (V) <-- (V) v (C)
    (C) <-- (op10)
    (op1n) <-- (op1n+1) [n=0...14]
    (op115) <-- (C)
    (count) <-- (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.

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 least significant bit 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)
Data Types	WORD	

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.

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) <-- (op115)
(op1n) <-- (op1n-1) [n=1...15]
(op10) <-- 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 most significant bit 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.

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 most significant bit 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

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.

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.

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.

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.

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	
Syntax	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.

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.

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
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.

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)	<-- (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.

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)	<-- (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.

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

3 - MAC INSTRUCTION SET

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

3.1 - Addressing modes

MAC instructions use some standard ST10 addressing modes such as GPR direct or #data₅ for immediate shift value. 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 pointer (GPR or IDX_i). The GPR pointer gives 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 combi-

nations of pointer post-modification for each of these 2 new addressing modes (see Table 27).

When using pointer post-modification addressing modes, the address pointed to (i.e the value in the IDX_i or Rw_n register) must be a legal address, even if its content is not modified. An odd value (e.g. in R0 when using [R0] post-modification addressing mode) will trigger the class-B hardware Trap 28h (Illegal Word Operand Access Trap (ILLOPA)).

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 (see Table 28).

Table 27 : Pointer post-modification for “[Rw_n⊗]” and “[IDX_i⊗]” addressing modes

Symbol	Mnemonic	Address Pointer Operation
“[IDX _i ⊗]” stands for ¹	[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	[Rw _n]	(Rw _n) <-- (Rw _n) (no-op)
	[Rw _n +]]	(Rw _n) <-- (Rw _n) +2 (n=0...15)
	[Rw _n -]	(Rw _n) <-- (Rw _n) -2 (n=0...15)
	[Rw _n + QR _j]	(Rw _n) <-- (Rw _n) + (QR _j) (n=0...15; j =0,1)
	[Rw _n - QR _j]	(Rw _n) <-- (Rw _n) - (QR _j) (n=0...15; j =0,1)

Note 1. IDX_i can only contain even values. Therefore, bit 0 always equals zero.

Table 28 : MAC register addresses for CoReg

Register	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

3.2 - 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 in Table 6, considering MAC instructions to be 4-byte instructions with a minimum state time number of 2.

3.3 - MAC instruction set summary

Table 29 : MAC instruction mnemonic by addressing mode and repeatability

Mnemonic	Addressing Modes	Rep	Mnemonic	Addressing Modes	Rep
CoMUL	Rw_n, Rw_m	No	CoMACM	$[IDX_i], [Rw_m]$	Yes
CoMULu	$[IDX_i], [Rw_m]$	No	CoMACMu		
CoMULus	$Rw_n, [Rw_m]$	No	CoMACMus		
CoMULsu			CoMACMsu		
CoMUL-			CoMACM-		
CoMULu-			CoMACMu-		
CoMULus-			CoMACMus-		
CoMULsu-			CoMACMsu-		
CoMUL + rnd			CoMACM + rnd		
CoMULu + rnd			CoMACMu + rnd		
CoMULus + rnd			CoMACMus + rnd		
CoMULsu + rnd			CoMACMsu + rnd		
CoMAC	Rw_n, Rw_m	No	CoMACMR		
CoMACu	$[IDX_i], [Rw_m]$	Yes	CoMACMRu		
CoMACus	$Rw_n, [Rw_m]$	Yes	CoMACMRus		
CoMACsu			CoMACMRsu		
CoMAC-			CoMACMR + rnd		
CoMACu-			CoMACMRu + rnd		
CoMACus-			CoMACMRus + rnd		
CoMACsu-			CoMACMRsu + rnd		
CoMAC + rnd			CoADD	Rw_n, Rw_m	No
CoMACu + rnd			CoADD2	$[IDX_i], [Rw_m]$	Yes
CoMACus + rnd			CoSUB	$Rw_n, [Rw_m]$	Yes
CoMACsu + rnd			CoSUB2		
CoMACR			CoSUBR		
CoMACRu			CoSUB2R		
CoMACRus			CoMAX		
CoMACRsu			CoMIN		
CoMACR + rnd			CoLOAD	Rw_n, Rw_m	No
CoMACRu + rnd			CoLOAD-	$[IDX_i], [Rw_m]$	No
CoMACRus + rnd			CoLOAD2	$Rw_n, [Rw_m]$	No
CoMACRsu + rnd			CoLOAD2-		
CoNOP	$[Rw_m]$	Yes	CoCMP		
	$[IDX_i], [Rw_m]$	Yes	CoSHL	Rw_n	Yes
CoNEG	-	No	CoSHR	#data ₅	No
CoNEG + rnd			CoASHR	$[Rw_m]$	Yes
CoRND			CoASHR + rnd		
CoSTORE	$Rw_n, CoReg$	No	CoABS	-	No
	$[Rw_n], CoReg$	Yes		Rw_n, Rw_m	No
CoMOV	$[IDX_i], [Rw_m]$	Yes		$[IDX_i], [Rw_m]$	No
				$Rw_n, [Rw_m]$	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 30 : MAC instruction function code (hexa)

Mnemonic	Function Code	Mnemonic	Function Code
CoMUL	C0	CoMACM	D8
CoMULu	00	CoMACMu	18
CoMULus	80	CoMACMus	98
CoMULsu	40	CoMACMsu	58
CoMUL-	C8	CoMACM-	E8
CoMULu-	08	CoMACMu-	28
CoMULus-	88	CoMACMus-	A8
CoMULsu-	48	CoMACMsu-	68
CoMUL + rnd	C1	CoMACM + rnd	D9
CoMULu + rnd	01	CoMACMu + rnd	19
CoMULus + rnd	81	CoMACMus + rnd	99
CoMULsu + rnd	41	CoMACMsu + rnd	59
CoMAC	D0	CoMACMR	F9
CoMACu	10	CoMACMRu	38
CoMACus	90	CoMACMRus	B8
CoMACsu	50	CoMACMRsu	78
CoMAC-	E0	CoMACMR + rnd	F9
CoMACu-	20	CoMACMRu + rnd	39
CoMACus-	A0	CoMACMRus + rnd	B9
CoMACsu-	60	CoMACMRsu + rnd	79
CoMAC + rnd	D1	CoADD	02
CoMACu + rnd	11	CoADD2	42
CoMACus + rnd	91	CoSUB	0A
CoMACsu + rnd	51	CoSUB2	4A
CoMACR	F0	CoSUBR	12
CoMACRu	30	CoSUB2R	52
CoMACRus	B0	CoMAX	3A
CoMACRsu	70	CoMIN	7A
CoMACR + rnd	F1	CoLOAD	22
CoMACRu + rnd	31	CoLOAD-	2A
CoMACRus + rnd	B1	CoLOAD2	62
CoMACRsu + rnd	71	CoLOAD2-	6A
CoNOP	5A	CoCMP	C2
CoNEG	32	CoSHL #data ₅	82
CoNEG + rnd	72	CoSHL other	8A
CoRND	B2	CoSHR #data ₅	92
CoABS -	1A	CoSHR other	9A
CoABS op1, op2	CA	CoASHR #data ₅	A2
CoSTORE	www:w000	CoASHR other	AA
CoMOV	00	CoASHR + rnd #data ₅	B2
		CoASHR + rnd other	BA

3.4 - MAC instruction conventions

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

3.4.1 - Operands

Operand	Description
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 opX (i.e. opX is used as pointer to the actual operand)
rnd	plus 00 0000 8000 _h

3.4.2 - Operations

Diadic operations	(opX)<-- (opY)	(opY) is MOVED into (opX)
	(opX) + (opY)	(opX) is ADDED to (opY)
	(opX) - (opY)	(opY) is SUBTRACTED from (opX)
	(opX) * (opY)	(opX) is MULTIPLIED by (opY)
	(opX) <--> (opY)	(opY) is COMPARED against (opX)
	opX\opY	(opX) is CONCATANATED to (opY) (LSW)
	Max ((opX), (opY))	MAXIMUM value between (opX) and (opY)
Min ((opX), (opY))	MINIMUM value between (opX) and (opY)	
Monadic Operations	(opX) <<	(opX) is Logically SHIFTED Left
	(opX) >>	(opX) is Logically SHIFTED Right
	(opX) >> _a	(opX) is Arithmetically SHIFTED Right
	Abs (opX)	ABSOLUTE value of (opX)

3.4.3 - Abbreviations

Abbreviation	Description
C	Carry flag in the MSW register
MP	MP mode in the MCW register
MS	MS mode in the MCW register
MAE	8 most significant bits of the accumulator (lowest byte of the MSW register)

3.4.4 - Data addressing Modes

Addressing mode	Description
"Rw _n ", or "Rw _m " :	General Purpose Registers (GPRs) where "n" and "m" are any value between 0 and 15.
[...]:	Indirect word memory location
CoReg :	MAC-Unit Register (MSW, MAH, MAL, MAS, MRW, MCW)
ACC :	MAC Accumulator consisting of (lowest byte of MSW)\MAH\MAL.
#data _x :	Immediate constant (the number of significant bits is represented by 'x').

3.4.5 - Instruction format

The instruction format is the same as that of the standard instruction set.

In addition, the following new symbols are used:

Instruction	Description
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.
www:w...	5-bit CoReg address for CoSTORE instructions.
ssss:	4-bit immediate shift value.
ssss:s...	5-bit immediate shift value.

Table 31 : 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

Table 31 : IDX Addressing Mode Encoding and GPR offset Encoding (continued)

Addressing Mode	4-bit Encoding
+ QR1	6 _h
- QR1	7 _h

3.4.6 - Flag states

Flag	Description
-	Unchanged
*	Modified

3.4.7 - 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 (MRW₁₂₋₀) + 1 times, therefore the maximum number of times an instruction can be repeated is 8 192 (2¹³) times.

#data₅ is an integer value specifying the number of times an instruction is repeated, #data₅ must be less than 32.

Therefore, CoXXX can only be repeated less than 32 times. When the MRW register is used in the repeat instruction, the 5-bit repeat field is set to 1.

3.4.8 - Shift value

The shifter authorizes only 8-bit left/right shifts. Shift values must be between 0-8 (inclusive).

3.5 - MAC instruction descriptions

Each instruction is described in a standard format. See "MAC instruction conventions" on page 126 for detailed information about the instruction conventions. The MAC instruction set is divided into 5 functional groups:

- Multiply and Multiply-Accumulate Instructions
- 40-bit Arithmetic Instructions
- Shift Instructions
- Compare Instructions
- Transfer Instructions

The instructions are described in alphabetical order.

CoABS	Absolute Value	
Group	40-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 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 $[IDX_i \otimes], [Rw_m \otimes]$	No	93 Xm CA 0:0qqq	4
CoABS $Rw_n, [Rw_m \otimes]$	No	83 nm CA 0:0qqq	4

CoADD(2)	Add	
Group	40-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 7FFF 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 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		Rep	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

ST10 FAMILY PROGRAMMING MANUAL

Examples

```

CoADD      R0, R1                ; (ACC) <-- (ACC) + (R1)\(R0)
CoADD2     R2, [R6+]             ; (ACC) <-- (ACC) + 2*( ((R6))\((R2) )
                                   ; (R6) <-- (R6) + 2

Repeat 3 times CoADD
CoADD      [IDX1+QX1], [R10+QR0] ; (ACC) <-- (ACC) + ( ((R10))\((IDX1)) )
                                   ; (R10) <-- (R10) + (QR0)
                                   ; (IDX1) <-- (IDX1) + (QX1)

Repeat MRW times CoADD2
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	-

CoASHR **Accumulator Arithmetic Shift Right with Optional Round**
Group Shift Instructions
Syntax CoASHRop1
 CoASHR op1, rnd
Operation (count) <-- (op1)
 (C) <-- 0
 DO WHILE (count) ≠ 0
 (ACC_n) <-- (ACC_{n+1}) [n=0-38]
 (count) <-- (count) -1
 END WHILE
 IF (rnd) THEN
 (ACC) <-- (ACC) + 00008000H
 (MAL) <-- 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 most significant bit was a 0 or with sign 1 if the original most significant bit was 1. Only shift values between 0 and 8 are allowed. "op1" can be either a 5-bit unsigned immediate data, or the least significant 5 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 7FFF FFFF_h or FF 8000 0000_h, respectively. This instruction is repeatable when "op 1" is not an immediate operand.

MAC 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 ₅	No	A3 00 A2 ssss:s000	4
CoASHR	#data ₅ , rnd	No	A3 00 B2 ssss:s000	4
CoASHR	[Rw _m ⊗]	Yes	83 mm AA rrrr:rqqq	4
CoASHR	[Rw _m ⊗], rnd	Yes	83 mm BA rrrr:rqqq	4

Examples

```
CoASHR     #3, rnd                      ; (ACC) <-- (ACC) >>a 3 + rnd
CoASHR     R3                           ; (ACC) <-- (ACC) >>a (R3)4-0
CoASHR     [R10 - QR0]                ; (ACC) <-- (ACC) >>a ((R10))4-0
                                         ; (R10) <-- (R10) - (QR0)
```



CoCMP	Compare	
Group	Compare Instructions	
Syntax	CoCMP	op1, op2
Operation	tmp (ACC)	<-- (op2)\(op1) <--> (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 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 ₁ ⊗], [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)

CoLOAD(2)(-)	Load Accumulator	
Group	40-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 7FFF FFFF_h or FF 8000 0000_h, respectively. This instruction is not repeatable and allows up to two parallel memory reads.

MAC 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



CoMAC(R/-)	Multiply-Accumulate & Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMAC op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1) * (op2)) << 1 (ACC) <-- (ACC) + (tmp) ELSE (tmp) <-- (op1) * (op2) (ACC) <-- (ACC) + (tmp) END IF </pre>
Syntax	CoMAC op1, op2, rnd
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1) * (op2)) << 1 (ACC) <-- (ACC) + (tmp) + 00 0000 8000_h ELSE (tmp) <-- (op1) * (op2) (ACC) <-- (ACC) + (tmp) + 00 0000 8000_h END IF (MAL) <-- 0 </pre>
Syntax	CoMAC- op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1) * (op2)) << 1 (ACC) <-- (ACC) - (tmp) ELSE (tmp) <-- (op1) * (op2) (ACC) <-- (ACC) - (tmp) END IF </pre>
Syntax	CoMACR op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1) * (op2)) << 1 (ACC) <-- (tmp) - (ACC) ELSE (tmp) <-- (op1) * (op2) (ACC) <-- (tmp) - (ACC) END IF </pre>
Syntax	CoMACRop1, op2, rnd
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1) * (op2)) << 1 (ACC) <-- (tmp) - (ACC) + 00 0000 8000_h ELSE (tmp) <-- (op1) * (op2) (ACC) <-- (tmp) - (ACC) + 00 0000 8000_h END IF (MAL) <-- 0 </pre>
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 the condition MP flag is set, it is one-bit left shifted, 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 “-” option is used to negate the specified product, the “R” option is used to negate the accumulator content, and finally the “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 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
CoMAC      [IDX1 - QX1], [R9+QR1] ; (ACC) <-- (ACC) + ((IDX1))*((R9))
                                     ; (R9) <-- (R9) + (QR1)
                                     ; (IDX1) <-- (IDX1) - (QX1)

Repeat MRW times CoMAC
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) + 00 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) + 00 0000 8000 _h (MAL) <-- 0
Data Types	DOUBLE WORD
Result	40-bit signed 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 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 ₁ ⊗], [Rw _m ⊗]	Yes	93 Xm 10 rrrr:rqqq	4
CoMACu-	[IDX ₁ ⊗], [Rw _m ⊗]	Yes	93 Xm 20 rrrr:rqqq	4
CoMACu	[IDX ₁ ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 11 rrrr:rqqq	4
CoMACRu	[IDX ₁ ⊗], [Rw _m ⊗]	Yes	93 Xm 30 rrrr:rqqq	4
CoMACRu	[IDX ₁ ⊗], [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)us(-)	Mixed Multiply-Accumulate & Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMACus op1, op2
Operation	(tmp) <-- (op1) * (op2) (ACC) <-- (ACC) + (tmp)
Syntax	CoMACus op1, op2, rnd
Operation	(tmp) <-- (op1) * (op2) (ACC) <-- (ACC) + (tmp) + 00 0000 8000 _h (MAL) <-- 0
Syntax	CoMACus- op1, op2
Operation	(tmp) <-- (op1) * (op2) (ACC) <-- (ACC) - (tmp)
Syntax	CoMACRus op1, op2
Operation	(tmp) <-- (op1) * (op2) (ACC) <-- (tmp) - (ACC)
Syntax	CoMACRus op1, op2, rnd
Operation	(tmp) <-- (op1) * (op2) (ACC) <-- (tmp) - (ACC) + 00 0000 8000 _h (MAL) <-- 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 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 90 00	4
CoMACus-	Rw _n , Rw _m	No	A3 nm A0 00	4
CoMACus	Rw _n , Rw _m , rnd	No	A3 nm 91 00	4
CoMACRus	Rw _n , Rw _m	No	A3 nm B0 00	4
CoMACRus	Rw _n , Rw _m , rnd	No	A3 nm B1 00	4
CoMACus	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 90 rrrr:rqqq	4
CoMACus-	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm A0 rrrr:rqqq	4
CoMACus	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 91 rrrr:rqqq	4
CoMACRus	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm B0 rrrr:rqqq	4
CoMACRus	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm B1 rrrr:rqqq	4
CoMACus	Rw _n , [Rw _m ⊗]	Yes	83 nm 90 rrrr:rqqq	4
CoMACus-	Rw _n , [Rw _m ⊗]	Yes	83 nm A0 rrrr:rqqq	4
CoMACus	Rw _n , [Rw _m ⊗], rnd	Yes	83 nm 91 rrrr:rqqq	4
CoMACRus	Rw _n , [Rw _m ⊗]	Yes	83 nm B0 rrrr:rqqq	4
CoMACRus	Rw _n , [Rw _m ⊗], rnd	Yes	83 nm B1 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)

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

Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMACsu	op1, op2
Operation	(tmp) (ACC)	<-- (op1) * (op2) <-- (ACC) + (tmp)
Syntax	CoMACsu	op1, op2, rnd
Operation	(tmp) (ACC) (MAL)	<-- (op1) * (op2) <-- (ACC) + (tmp) + 00 0000 8000 _h <-- 0
Syntax	CoMACsu-	op1, op2
Operation	(tmp) (ACC)	<-- (op1) * (op2) <-- (ACC) - (tmp)
Syntax	CoMACRsu	op1, op2
Operation	(tmp) (ACC)	<-- (op1) * (op2) <-- (tmp) - (ACC)
Syntax	CoMACRsu	op1, op2, rnd
Operation	(tmp) (ACC) (MAL)	<-- (op1) * (op2) <-- (tmp) - (ACC) + 00 0000 8000 _h <-- 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 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 50 00	4
CoMACsu-	Rw _n , Rw _m	No	A3 nm 60 00	4
CoMACsu	Rw _n , Rw _m , rnd	No	A3 nm 51 00	4
CoMACRsu	Rw _n , Rw _m	No	A3 nm 70 00	4
CoMACRsu	Rw _n , Rw _m , rnd	No	A3 nm 71 00	4
CoMACsu	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 50 rrrr:rqqq	4
CoMACsu-	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 60 rrrr:rqqq	4
CoMACsu	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 51 rrrr:rqqq	4
CoMACRsu	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 70 rrrr:rqqq	4
CoMACRsu	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 71 rrrr:rqqq	4
CoMACsu	Rw _n , [Rw _m ⊗]	Yes	83 nm 50 rrrr:rqqq	4
CoMACsu-	Rw _n , [Rw _m ⊗]	Yes	83 nm 60 rrrr:rqqq	4
CoMACsu	Rw _n , [Rw _m ⊗], rnd	Yes	83 nm 51 rrrr:rqqq	4
CoMACRsu	Rw _n , [Rw _m ⊗]	Yes	83 nm 70 rrrr:rqqq	4
CoMACRsu	Rw _n , [Rw _m ⊗], rnd	Yes	83 nm 71 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)
CoMACRsu	[IDX1 - QX0], [R4], rnd	; (ACC) <-- ((IDX1))*((R4)) - (ACC) ; (IDX1) <-- (IDX1) - (QX0)

CoMACM(R/-)	Multiply-Accumulate Parallel Data Move & Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMACM op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1))*((op2)) << 1 (ACC) <-- (ACC) + (tmp) ELSE (tmp) <-- ((op1))*((op2)) (ACC) <-- (ACC) + (tmp) END IF ((IDX_i(-⊗))) <-- ((IDX_i)) </pre>
Syntax	CoMACM op1, op2, rnd
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1))*((op2)) << 1 (ACC) <-- (ACC) + (tmp) + 00 0000 8000_h ELSE (tmp) <-- ((op1))*((op2)) (ACC) <-- (ACC) + (tmp) + 00 0000 8000_h END IF (MAL) <-- 0 ((IDX_i(-⊗))) " ((IDX_i)) </pre>
Syntax	CoMACM- op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1))*((op2)) << 1 (ACC) <-- (ACC) - (tmp) ELSE (tmp) <-- ((op1))*((op2)) (ACC) <-- (ACC) - (tmp) END IF ((IDX_i(-⊗))) <-- ((IDX_i)) </pre>
Syntax	CoMACMR op1, op2
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1))*((op2)) << 1 (ACC) <-- (tmp) - (ACC) ELSE (tmp) <-- ((op1))*((op2)) (ACC) <-- (tmp) - (ACC) END IF ((IDX_i(-⊗))) <-- ((IDX_i)) </pre>
Syntax	CoMACMR op1, op2, rnd
Operation	<pre> IF (MP = 1) THEN (tmp) <-- ((op1))*((op2)) << 1 (ACC) <-- (tmp) - (ACC) + 00 0000 8000_h ELSE (tmp) <-- ((op1))*((op2)) (ACC) <-- (tmp) - (ACC) + 00 0000 8000_h END IF (MAL) <-- 0 ((IDX_i(-⊗))) <-- ((IDX_i)) </pre>
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 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)
$[IDX_i+]$	$(IDX_i) - 2$
$[IDX_i-]$	$(IDX_i) + 2$
$[IDX_i+QX_j]$	$(IDX_i) - (QX_j)$
$[IDX_i-QX_j]$	$(IDX_i) + (QX_j)$

MAC 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	Yes	93 Xm D8 rrrr:rqqq	4
CoMACM-	Yes	93 Xm E8 rrrr:rqqq	4
CoMACM	Yes	93 Xm D9 rrrr:rqqq	4
CoMACMR	Yes	93 Xm F8 rrrr:rqqq	4
CoMACMR	Yes	93 Xm F9 rrrr:rqqq	4

Examples

```
CoMACM [IDX1+QX0],[R10+QR1], rnd ; (ACC) <-- (ACC) + ((IDX1))*((R10)) + rnd
; (R10) <-- (R10) + (QR1)
; ( ((IDX1)-(QX0)) ) <-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX0)
```

Repeat 3 times CoMACM

```
CoMACM [IDX0 - QX0], [R8+QR0] ; (ACC) <-- (ACC) + ((IDX0))*((R8))
; (R8) <-- (R8) + (QR0)
; ( ((IDX0) + (QX0)) ) <-- ((IDX0))
; (IDX0) <-- (IDX0) - (QX0)
```

Repeat MRW times CoMACM

```
CoMACM [IDX1+QX1], [R7 - QR0] ; (ACC) <-- (ACC) - ((IDX1))*((R7))
; (R7) <-- (R7) - (QR0)
; ( ((IDX1) - (QX1)) ) <-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX1)
```

CoMACM(R)u(-)	Unsigned Multiply-Accumulate Parallel Data Move & Optional Round	
Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMACMu	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMu	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp) + 00 0000 8000 _h
	(MAL)	<-- 0
	(IDX _i (-⊗))	<-- ((IDX _i))
Syntax	CoMACMu-	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) - (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRu	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRu	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC) + 00 0000 8000 _h
	(MAL) <-- 0	
	((IDX _i (-⊗)))	<-- ((IDX _i))
Data Types	DOUBLE WORD	
Result	40-bit signed value	

Description

Multiplies the two signed 16-bit source operands “op1” and “op2”. The unsigned 32-bit product is first zero-extended, 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 -QX _j]	(IDX _i) + (QX _j)

MAC 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)+ ((IDX1)) * ((R10))+ rnd
; (R10) <-- (R10) + (QR1)
; ( ((IDX1) - (QX0)) ) <-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX0)
```

Repeat 3 times CoMACMu

```
CoMACMu [IDX0 - QX0], [R8+QR0] ; (ACC) <-- (ACC) + ((IDX0))*((R8))
; (R8) <-- (R8) + (QR0)
; ( ((IDX0) + (QX0)) ) <-- ((IDX0))
; (IDX0) <-- (IDX0) - (QX0)
```

Repeat MRW times CoMACMRu

```
CoMACMRu [IDX1+QX1], [R7 - QR0] ; (ACC) <-- ((IDX1))*((R7)) - (ACC)
; (R7) <-- (R7) - (QR0)
; ( ((IDX1) - (QX1)) ) <-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX1)
```

CoMACM(R)us(-)	Mixed Multiply-Accumulate Parallel Data Move & Optional Round	
Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMACMus	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMus	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp) + 00 0000 8000 _h
	(MAL)	<-- 0
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMus-	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) - (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRus	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRus	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC) + 00 0000 8000 _h
	(MAL)	<-- 0
	((IDX _i (-⊗)))	<-- ((IDX _i))
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, 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 - QX _j]	(IDX _i) + (QX _j)

MAC 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 98 rrrr:rqqq	4
CoMACMus-	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm A8 rrrr:rqqq	4
CoMACMus	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 99 rrrr:rqqq	4
CoMACMRus	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm B8 rrrr:rqqq	4
CoMACMRus	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm B9 rrrr:rqqq	4

Examples

```
CoMACMus [IDX1+QX0], [R10+QR1], rnd ; (ACC)<--(ACC) + ((IDX1))*((R10)) +rnd
; (R10) <-- (R10) + (QR1)
; ( ((IDX1) - (QX0)) )<-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX0)
```

Repeat 3 times CoMACMus

```
CoMACMus [IDX0 - QX0], [R8+QR0] ; (ACC) <-- (ACC) + ((IDX0))*((R8))
; (R8) <-- (R8) + (QR0)
; ( ((IDX0) + (QX0)) ) <-- ((IDX0))
; (IDX0) <-- (IDX0) - (QX0)
```

Repeat MRW times CoMACMRus

```
CoMACMRus [IDX1+QX1], [R7 - QR0], rnd ; (ACC)<--((IDX1))*((R7))-(ACC)+rnd
; (R7) <-- (R7) - (QR0)
; ( ((IDX1) - (QX1)) )<-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX1)
```

CoMACM(R)su(-)	Mix. Multiply-Accumulate Parallel Data Move & Optional Round	
Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMACMsu	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMsu	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) + (tmp) + 00 0000 8000 _h
	(MAL)	<-- 0
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMsu-	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (ACC) - (tmp)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRsu	op1, op2
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC)
	((IDX _i (-⊗)))	<-- ((IDX _i))
Syntax	CoMACMRsu	op1, op2, rnd
Operation	(tmp)	<-- ((op1))*((op2))
	(ACC)	<-- (tmp) - (ACC) + 00 0000 8000 _h
	(MAL)	<-- 0
	((IDX _i (-⊗)))	<-- ((IDX _i))
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, 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 - QX _j]	(IDX _i) + (QX _j)

MAC Flags

N	Z	C	SV	E	SL
*	*	*	*	*	*

- N Set if the m.s.b. 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 58 rrrr:rqqq	4
CoMACMsu-	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 68 rrrr:rqqq	4
CoMACMsu	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 59 rrrr:rqqq	4
CoMACMRsu	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 78 rrrr:rqqq	4
CoMACMRsu	[IDX _i ⊗], [Rw _m ⊗], rnd	Yes	93 Xm 79 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

```
CoMACMsu [IDX0 - QX0], [R8+QR0], rnd ; (ACC) <-- (ACC) + ((IDX0))*((R8))
; (R8) <-- (R8) + (QR0)
; ( ((IDX0) + (QX0)) ) <-- ((IDX0))
; (IDX0) <-- (IDX0) - (QX0)
```

Repeat MRW times CoMACMRsu

```
CoMACMRsu [IDX1+QX1], [R7 - QR0], rnd ; (ACC) <-- ((IDX1))*((R7)) - (ACC) + rnd
; (R7) <-- (R7) - (QR0)
; ( ((IDX1)) - (QX1)) ) <-- ((IDX1))
; (IDX1) <-- (IDX1) + (QX1)
```


CoMAX	Maximum
Group	Compare Instructions
Syntax	CoMAXop1, 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 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, R10                    ; (ACC) <-- Max( (ACC), (R10)\(R1) )

Repeat 23 times CoMAX

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 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

CoMIN    R5, [R6 - QR0]           ; (ACC) <-- min( (ACC), ((R6))\((R5)) )
                                           ; (R6) <-- (R6) - (QR0)
    
```

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, IDX_i can address the entire memory. This instruction does not affect the Mac Flags but modify the CPU Flags as any other MOV instruction.

CPU 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 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)
```

CoMUL(-)	Signed Multiply & Optional Round
Group	Multiply/Multiply-Accumulate Instructions
Syntax	CoMUL op1, op2
Operation	<pre> IF (MP = 1) THEN (ACC) <-- ((op1) * (op2)) << 1 ELSE (ACC) <-- (op1) * (op2) END IF </pre>
Syntax	CoMUL- op1, op2
Operation	<pre> IF (MP = 1) THEN (ACC) <-- - (((op1) * (op2)) << 1) ELSE (ACC) <-- - ((op1) * (op2)) END IF </pre>
Syntax	CoMUL op1, op2, rnd
Operation	<pre> IF (MP = 1) THEN (ACC) <-- ((op1) * (op2)) << 1 + 00 0000 8000_h ELSE (ACC) <-- (op1) * (op2) + 00 0000 8000_h END IF (MAL) <-- 0 </pre>
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 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	$[IDX_i \otimes], [Rw_m \otimes]$	No	93 Xm C0 0:0qqq	4
CoMUL-	$[IDX_i \otimes], [Rw_m \otimes]$	No	93 Xm C8 0:0qqq	4
CoMUL	$[IDX_i \otimes], [Rw_m \otimes], rnd$	No	93 Xm C1 0:0qqq	4
CoMUL	$Rw_n, [Rw_m \otimes]$	No	83 nm C0 0:0qqq	4
CoMUL-	$Rw_n, [Rw_m \otimes]$	No	83 nm C8 0:0qqq	4
CoMUL	$Rw_n, [Rw_m \otimes], rnd$	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)	<-- (op1) * (op2)
Syntax	CoMULu-	op1, op2
Operation	(ACC)	<-- - ((op1) * (op2))
Syntax	CoMULu	op1, op2, rnd
Operation	(ACC)	<-- (op1) * (op2) + 00 0000 8000 _h
	(MAL)	<-- 0
Data Types	DOUBLE WORD	
Result	32-bit signed 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 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

Notes: 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))*((R9)) + 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=x	FFFF _h	FFFF _h	0	00 _h	FFFE _h	0001 _h	0	0	0	-	0	-
			1	00 _h	FFFE _h	0000 _h	0	0	0	-	0	-

CoMULus(-)	Mixed Multiply & Optional Round	
Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMULus	op1, op2
Operation	(ACC)	<-- (op1) * (op2)
Syntax	CoMULus-	op1, op2
Operation	(ACC)	<-- - ((op1) * (op2))
Syntax	CoMULus	op1, op2, rnd
Operation	(ACC)	<-- (op1) * (op2) + 00 0000 8000 _h
	(MAL)	<-- 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 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 80 00	4
CoMULus-	Rw _n , Rw _m	No	A3 nm 88 00	4
CoMULus	Rw _n , Rw _m , rnd	No	A3 nm 81 00	4
CoMULus	[IDX _i ⊗], [Rw _m ⊗]	No	93 Xm 80 0:0qqq	4
CoMULus-	[IDX _i ⊗], [Rw _m ⊗]	No	93 Xm 88 0:0qqq	4
CoMULus	[IDX _i ⊗], [Rw _m ⊗], rnd	No	93 Xm 81 0:0qqq	4
CoMULus	Rw _n , [Rw _m ⊗]	No	83 nm 80 0:0qqq	4
CoMULus-	Rw _n , [Rw _m ⊗]	No	83 nm 88 0:0qqq	4
CoMULus	Rw _n , [Rw _m ⊗], rnd	No	83 nm 81 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))*((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	FA2A _h	F456 _h	1	0	0	-	0	-
			1	FF _h	FA2B _h	0000 _h	1	0	0	-	0	-

CoMULsu(-)	Mixed Multiply & Optional Round	
Group	Multiply/Multiply-Accumulate Instructions	
Syntax	CoMULsu	op1, op2
Operation	(ACC)	<-- (op1) * (op2)
Syntax	CoMULsu-	op1, op2
Operation	(ACC)	<-- - ((op1) * (op2))
Syntax	CoMULsu	op1, op2, rnd
Operation	(ACC)	<-- (op1) * (op2) + 00 0000 8000 _h
	(MAL)	<-- 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 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 40 00	4
CoMULsu-	Rw _n , Rw _m	No	A3 nm 48 00	4
CoMULsu	Rw _n , Rw _m , rnd	No	A3 nm 41 00	4
CoMULsu	[IDX _i ⊗], [Rw _m ⊗]	No	93 Xm 40 0:0qqq	4
CoMULsu-	[IDX _i ⊗], [Rw _m ⊗]	No	93 Xm 48 0:0qqq	4
CoMULsu	[IDX _i ⊗], [Rw _m ⊗], rnd	No	93 Xm 41 0:0qqq	4
CoMULsu	Rw _n , [Rw _m ⊗]	No	83 nm 40 0:0qqq	4
CoMULsu-	Rw _n , [Rw _m ⊗]	No	83 nm 48 0:0qqq	4
CoMULsu	Rw _n , [Rw _m ⊗], rnd	No	83 nm 41 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	-

CoNEG **Negate Accumulator with Optional Rounding**

Group 32-bit Arithmetic Instructions

Syntax CoNEG
CoNEG nd

Operation IF (rnd) THEN
 (ACC) <-- 0 - (ACC) + 00 0000 8000_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 7FFF FFFF_h or FF 8000 0000_h, respectively. This instruction is not repeatable

MAC Flags

N	Z	C	SV	E	SL
*	*	*	*	*	*

- N Set if the m.s.b. 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	-

CoNOP	No-Operation
Group	40-bit Arithmetic Instructions
Syntax	CoNOP
Operation	No Operation

Description

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

MAC 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 1m 5A rrrr:rqqq	4
CoNOP	[IDX _i ⊗], [Rw _m ⊗]	Yes	93 Xm 5A rrrr:rqqq	4

Example

CoNOP	[IDX0+QX1], [R11+QR1]	; (R11) <-- (R11) + (QR1) ; (IDX0) <-- (IDX0) + (QX1)
-------	-----------------------	--

ST10 FAMILY PROGRAMMING MANUAL

CoRND	Round Accumulator
Group	Shift Instructions
Syntax	CoRND
Operation	(ACC) <-- (ACC) + 00 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 7FFF FFFF_h or FF 8000 0000_h, respectively. This instruction is not repeatable.

MAC 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

CoSHL	Accumulator Logical Shift Left	
Group	Shift Instructions	
Syntax	CoSHL	op1
Operation	(count)	<-- (op1)
	(C)	<-- 0
	DO WHILE (count) ≠ 0	
	(C)	<-- (ACC ₃₉)
	(ACC _n)	<-- (ACC _{n-1}) [n=1...39]
	(ACC ₀)	<-- 0
	(count)	<-- (count) -1
	END WHILE	
Data types	ACCUMULATOR	
Result	40-bit signed value	

Description

Shifts the ACC register left by the number of times specified by the operand op1. The least significant bits of the result are filled with zeros. Only shift values from 0 to 8 (inclusive) are allowed. "op1" can be either a 5-bit unsigned immediate data, or the least significant 5 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 7FFF FFFF_h or FF 8000 0000_h, respectively. This instruction is repeatable when "op1" is not an immediate operand.

MAC 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 Carry flag is set according to the last most significant bit shifted out of ACC.
- 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 ₅	No	A3 00 82 ssss:s000	4
CoSHL	[Rw _m ⊗]	Yes	83 mm 8A rrrr:rqqq	4

Examples

- CoSHL #3 ; (ACC) <-- (ACC) << 3
- CoSHL R3 ; (ACC) <-- (ACC) << (R3)₄₋₀
- CoSHL [R10 - QR0] ; (ACC) <-- (ACC) << ((R10))₄₋₀
- ; (R10) <-- (R10) - (QR0)



CoSHR	Accumulator Logical Shift Right	
Group	Shift Instructions	
Syntax	CoSHR	op1
Operation	(count)	<-- (op1)
	(C)	<-- 0
	DO WHILE (count) ≠ 0	
	((ACC _n)	<-- (ACC _{n+1}) [n=0-38]
	(ACC ₃₉)	<-- 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 5-bit unsigned immediate data, or the least significant 5 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 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	Rep	Format	Bytes
CoSHR Rw _n	Yes	A3 nn 9A rrrr:r000	4
CoSHR #data ₅	No	A3 00 92 ssss:s000	4
CoSHR [Rw _m ⊗]	Yes	83 mm 9A rrrr:rqqq	4

Examples

CoSHR	#3	;	(ACC) <-- (ACC) >> 3
CoSHR	R3	;	(ACC) <-- (ACC) >> (R3) ₄₋₀
CoSHR	[R10 - QR0]	;	(ACC) <-- (ACC) >> ((R10)) ₄₋₀
		;	(R10) <-- (R10) - (QR0)

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 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 such as MSW, MAH, MAL, MAS, MRW or MCW. In this case, a NOP must be inserted between the CoSTORE and MOV instruction.

Examples

```
CoSTORE    [R11+QR1], MAS    ; ((R11)) <-- limited((ACC))
                                     ; (R11) <-- (R11) + (QR1)

Repeat 3 times CoSTORE
CoSTORE    [R2-], MAL        ; ((R2)) <-- (MAL)
                                     ; (R2) <-- (R2) - 2
```

CoSUB(2)(R)	Subtract	
Group	Arithmetic Instructions	
Syntax	CoSUB	op1, op2
Operation	(tmp) (ACC)	<-- (op2)\(op1) <-- (ACC) - (tmp)
Syntax	CoSUB2	op1, op2
Operation	(tmp) (ACC)	<-- 2 * (op2)\(op1) <-- (ACC) - (tmp)
Syntax	CoSUBR	op1, op2
Operation	(tmp) (ACC)	<-- (op2)\(op1) <-- (tmp) - (ACC)
Syntax	CoSUB2R	op1, op2
Operation	(tmp) (ACC)	<-- 2 * (op2)\(op1) <-- (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 the “R” option is used, and stores 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. The “2” option indicates that the 40-bit operand is also multiplied by 2, prior to being subtracted/added from/to the ACC/negated ACC. When the most significant bit of the MCW register is set and when a 32-bit overflow or underflow occurs, the obtained result becomes 00 7FFF 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 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 times CoSUB
CoSUB          [IDX1+QX1], [R10+QR0] ; (ACC) <-- (ACC) - ( (R10)\((IDX1)) )
                                   ; (R10) <-- (R10) + (QR0)
                                   ; (IDX1) <-- (IDX1) + (QX1)

Repeat MRW times CoSUB2R
CoSUB2R       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	-
CoSUB2R	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

4 - REVISION HISTORY

Revision 5 - version 1 of January 2000

Chapter 2.1.4

See 1: $GPRAddress = (CP + 2 \times ShortAddress)$

See 3: $LongAddress = (GPRAddress) + Constant$

See 4: $PhysicalAddress = (DPPi) + LongAddress \wedge 3FFFh$

See5: $(GPRPAddress) = (GPRDAddress) + \Delta$

Chapter 2.2.3 Additional State Times:

"Jumps into the internal ROM Space :..."

– Label

– $I_n + 1$

– $I_n + 2$ JMPR cc_NC, label

Chapter 2.4:

Table 9, 10, 11, 12, 13 , 14, 15, 16, 17, 18, 19,

All column 16 bit N-MUX, 16 bit MUX, 8 bit N-MUX, 8 bit MUX.

Revision A - revision 4

This document number 7096626A is the transfer onto ADCS of document 42-1735-05 on the Bristol document control system. This revision includes extensive modifications to format. The major modifications to content are summarized in this table:

r -> R	In MAC instructions, upper case R has replaced lower case r for Reverse operation.
#data ₄ -> #data ₅	In MAC instructions, immediate shift value uses 5 bits to be coded, not 4.
Table 30 Instr. CoMACMus Instr. CoMACMus- Instr. CoMACMus rnd Instr. CoMACMR	function code is 98 function code is A8 function code is 99 function code is F9
Instr. CoMACM(R)su(-) Addressing Mode CoMACRsu [IDX _i ⊗], [Rw _m ⊗] CoMACRsu [IDX _i ⊗], [Rw _m ⊗], rnd CoMACRsu Rw _n , [Rw _m ⊗], rnd	93 Xm 70 rrrr:rqqq 93 Xm 71 rrrr:rqqq 93 Xm 71 rrrr:rqqq
correction in Multiplication examples CoMULu(-) and coMULus(-)	
Instruction BMOV	flag Z corrected
Instruction BMOVN	flag Z corrected
Instruction JNBS	flag Z corrected
Instruction MUL	flag N corrected
Instruction MULU	flag N corrected
Instruction SUBCB	flag Z corrected

Revision 4 - revision 3

Instructions: CoMULsu(-), CoMULus(-), CoMAC(r)su(-), CoMAC(r)us(-), CoMACM(r)su(-), CoMAC(r)us(-), CoNOP, CoSHL, CoSHR, CoASHR, CoSTORE	Addressing modes corrected. Function code in Table 30 corrected.
Instructions JBC and JNBS:	Condition flags corrected.
Table 22: <i>Instruction set ordered by Hex code</i> :	Updated to include section C0-FF, MAC instructions and working register indexes.
Instruction CoMULus(-):	Example corrected.
Table 5: <i>Branch target address summary</i> :	Seg address range corrected.
Table 24: <i>Condition codes</i> :	Condition Code Mnemonic cc_N corrected.
Section 2.4.6: <i>Repeated instruction syntax</i> :	Sentence added.
Instruction CoSHL:	Description clarified: "Only shift values from 0 to 8 (inclusive)".
Instruction CoNOP:	[IDX _i ⊗] addressing mode and example removed. Reference to this addressing mode removed from Table 29.
Instruction BCLR:	Condition flag Z corrected.
MAC instruction descriptions:	Ordered Alphabetically.
Section 2.1: <i>Addressing modes</i> :	Paragraph added.
Section 1.2.1: <i>Definition of measurement units</i> :	[Fcpu] changed to 0-50MHz.

Revision 3 - revision 2

CoSUB2r replaced CoSUBr2.

In MAC instructions, lower case r has replaced upper case R for optional repeat.

Revision 2 - revision 1

"Definition of measurement units" on page 12, ALE Cycle Time corrected.

"Integer Addition with Carry" on page 59: instruction name changed from ADDBC to ADDCB.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics 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 STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics

© 2000 STMicroelectronics - All Rights Reserved

STMicroelectronics GROUP OF COMPANIES

Australia - Brazil - Canada - China - France - Germany - Italy - Japan - Korea - Malaysia - Malta - Mexico - Morocco
The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - Thailand - United Kingdom - U.S.A.

<http://www.st.com>