2 - STANDARD INSTRUCTION SET

2.1 - Addressing Modes

2.1.1 - Short adressing 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 spacePhysicalAddress = BaseAddress + Δ x ShortAddress.

Note: Δ = 1 for byte GPRs, Δ = 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 00'F000h (CP) (CP)	+ 2*reg + 2*reg + 2*(reg^0Fh) + 1*(reg^0Fh)	reg reg reg reg	= 00h...EFh = 00h...EFh = F0h...FFh = F0h...FFh	SFRs ESFRs GPRs GPRs	(Word, Low byte) (Word, Low byte) (Word) 16 values (Bytes) 16 values
bitoff	00'FD00h 00'FF00h (CP)	+ 2*bitoff + 2*(bitoff^FFh) + 2*(bitoff^0Fh)	bitoff bitoff bitoff	= 00h...7Fh = 80h...EFh = F0h...FFh	RAM SFR GPR	Bit word offset 128 values Bit word offset 128 values Bit word offset 16 values
bitaddr	Word offset as with bitoff Immediate bit position		bitoff bitpos	= 00h...FFh = 0...15	Any single bit	

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)**.**

GPRAddress = (CP) + 2 x ShortAddress

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

(GPRAddress) = (GPRAddress) - Δ [optional step!]

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

Long Address = (GPR Address) + Constant

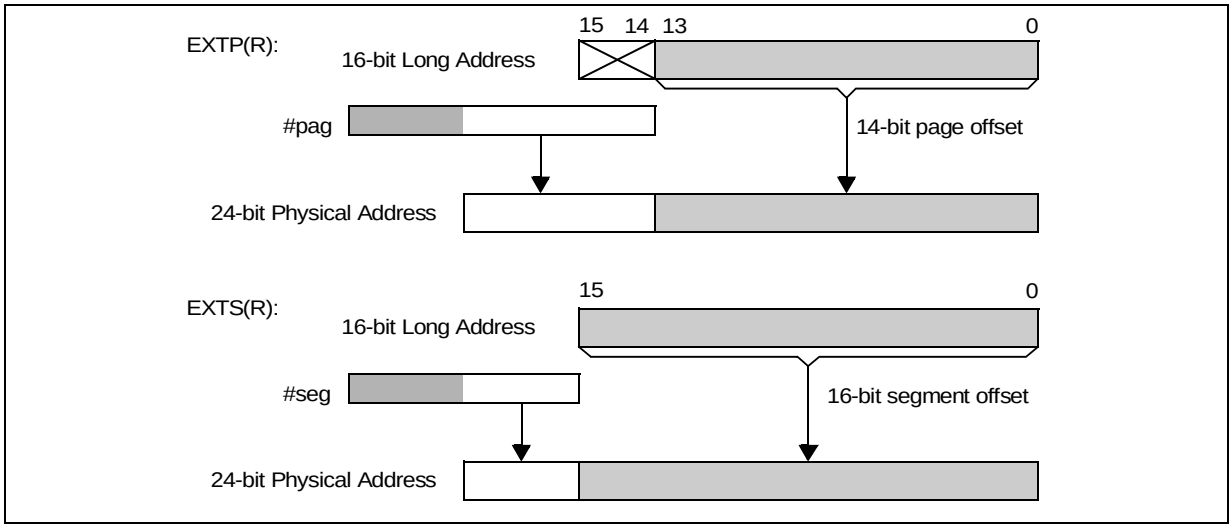
4. Calculate the physical 18-bit or 24-bit address using the resulting long address and the corresponding DPP register content (see long 'mem' addressing modes).

Physical Address = (DPPi) + Long Address^3FFFh

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

(GPR Address) = (GPR Address) + Δ [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 bytewide immediate constants.

For optimum utilization of the available code storage, these constants are represented in the instruction formats by either 3, 4, 8 or 16 bits.

Therefore, short constants are always zero-extended, while long constants can be truncated 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

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
MOVBZ	reg, mem	4	ATOMIC	$\#data_2$	2
	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 RET RETI RETS	-	2	SRST/IDLE PWRDN SRVWDT DISWDT EINIT	-	4

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").

$\#mask_8$: 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.

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
“0”	The flag is cleared by the operation.
“NOR”	The flag contains the logical NORing of the two specified bit operands.
“AND”	The flag contains the logical ANDing of the two specified bit operands.
“'OR”	The flag contains the logical ORing of the two specified bit operands.
“XOR”	The flag contains the logical XORing of the two specified bit operands.
“B”	The flag contains the original value of the specified bit operand.
“$\overline{B}$”	The flag contains the complemented value of the specified bit operand

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 (#pag10)
MM MM	16-bit address (mem or caddr; low byte, high byte)
## ##	16-bit immediate constant (#$data_{16}$; low byte, high byte)

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

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

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

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

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

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