B. Instruction Set Desciption

The MSP430 Core CPU architecture evolved from the idea of using a reduced instruction set with highly transparent instruction formats. There are core instructions that are implemented into hardware, and emulated instructions that use the hardware construction and emulate instructions with high efficiency. The emulated instructions use core instructions with the additional built-in constant generators CG1 and CG2. Both the core instructions and the emulated instructions are described in this section. The mnemonics of the emulated instructions are used with the examples.

The words in program memory used by an instruction vary from 1 to 3 words, depending on the combination of addressing modes.
Each instruction uses a minimum of one word (two bytes) in the program memory. The indexed, symbolic, absolute and immediate modes need one additional word in the program memory. These four modes are available for the source operand. The indexed, symbolic and absolute mode can be used for the destination operand.
The instruction combination for source and destination consumes one to three words of code memory.

Instruction Set Overview

				Status Bits			
				V	N	Z	C
*	ADC[.W];ADC.B	dst	dst + C -> dst	*	*	*	*
	ADD[.W];ADD.B	src,dst	src + dst -> dst	*	*	*	*
	ADDC[.W];ADDC.B	src,dst	src + dst + C -> dst	*	*	*	*
	AND[.W];AND.B	src,dst	src .and. dst -> dst	0	*	*	*
	BIC[.W];BIC.B	src,dst	.not.src .and. dst -> dst	-	-	-	-
	BIS[.W];BIS.B	src,dst	src .or. dst -> dst	-	-	-	-
	BIT[.W];BIT.B	src,dst	src .and. dst	0	*	*	*
*	BR	dst	Branch to	-	-	-	-
	CALL	dst	PC+2 -> stack, dst -> PC	-	-	-	-
*	CLR[.W];CLR.B	dst	Clear destination	-	-	-	-
*	CLRC	Clear carry bit		-	-	-	0
*	CLRN	Clear negative bit		-	0	-	-
*	CLRZ	Clear zero bit		-	-	0	-
	CMP[.W];CMP.B	src,dst	dst - src	*	*	*	*
*	DADC[.W];DADC.B	dst	dst + C -> dst (decimal)	*	*	*	*
	DADD[.W];DADD.B	src,dst	src + dst + C -> dst (decimal)	*	*	*	*
*	DEC[.W];DEC.B	dst	dst - 1 -> dst	*	*	*	*
*	DECD[.W];DECD.B	dst	dst - 2 -> dst	*	*	*	*
*	DINT	Disable interrupt		-	-	-	-
*	EINT	Enable interrupt		-	-	-	-
*	INC[.W];INC.B	dst	Increment destination, dst +1 -> dst	*	*	*	*
*	INCD[.W];INCD.B	dst	Double-Increment destination, dst+2->dst	*	*	*	*
*	INV[.W];INV.B	dst	Invert destination	*	*	*	*
	JC/JHS	Label	Jump to Label if Carry-bit is set	-	-	-	-
	JEQ/JZ	Label	Jump to Label if Zero-bit is set	-	-	-	-
	JGE	Label	Jump to Label if (N .XOR. V) = 0	-	-	-	-
	JL	Label	Jump to Label if (N .XOR. V) = 1	-	-	-	-
	JMP	Label	Jump to Label unconditionally	-	-	-	-
	JN	Label	Jump to Label if Negative-bit is set	-	-	-	-
	JNC/JLO	Label	Jump to Label if Carry-bit is reset	-	-	-	-
	JNE/JNZ	Label	Jump to Label if Zero-bit is reset	-	-	-	-

Note: Marked instructions are emulated instructions

All marked instructions (*) are emulated instructions. The emulated instructions use core instructions combined with the architecture and implementation of the CPU, for higher code efficiency and faster execution.

				Status Bits			
				V	N	Z	C
	MOV[.W];MOV.B	src,dst	src -> dst	-	-	-	-
*	NOP		No operation	-	-	-	-
*	POP[.W];POP.B	dst	Item from stack, SP+2 → SP	-	-	-	-
	PUSH[.W];PUSH.B	src	SP - 2 → SP, src → @SP	-	-	-	-
	RETI		Return from interrupt TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SZP	*	*	*	*
*	RET		Return from subroutine TOS → PC, SP + 2 → SP	-	-	-	-
*	RLA[.W];RLA.B	dst	Rotate left arithmetically	*	*	*	*
*	RLC[.W];RLC.B	dst	Rotate left through carry	*	*	*	*
	RRA[.W];RRA.B	dst	MSB → MSB → ....LSB → C	0	*	*	*
	RRC[.W];RRC.B	dst	C → MSB → .....LSB → C	*	*	*	*
*	SBC[.W];SBC.B	dst	Subtract carry from destination	*	*	*	*
*	SETC		Set carry bit	-	-	-	1
*	SETN		Set negative bit	-	1	-	-
*	SETZ		Set zero bit	-	-	1	-
	SUB[.W];SUB.B	src,dst	dst + .not.src + 1 → dst	*	*	*	*
	SUBC[.W];SUBC.B	src,dst	dst + .not.src + C → dst	*	*	*	*
	SWPB	dst	swap bytes	-	-	-	-
	SXT	dst	Bit7 → Bit8 ..... Bit15	0	*	*	*
*	TST[.W];TST.B	dst	Test destination	0	*	*	1
	XOR[.W];XOR.B	src,dst	src .xor. dst → dst	*	*	*	*

Note: Marked instructions

All marked instructions (*) are emulated instructions. The emulated instructions use core instructions combined with the architecture and implementation of the CPU, for higher code efficiency and faster execution.

B

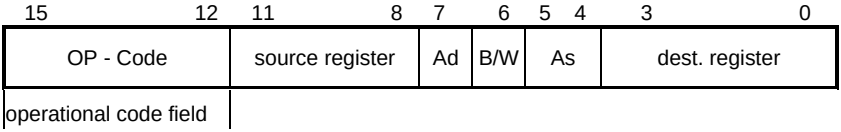
Instruction Formats

Double operand instructions (core instructions)

The instruction format using double operands consists of four main fields, in total a 16bit code:

- operational code field, 4bit [OP-Code]
- source field, 6bit [source register + As]
- byte operation identifier, 1bit [BW]
- destination field, 5bit [dest. register + Ad]

The source field is composed of two addressing bits and the 4bit register number (0....15); the destination field is composed of one addressing bit and the 4bit register number (0....15). The byte identifier B/W indicates whether the instruction is executed as a byte (B/W=1) or as a word instruction (B/W=0)



				Status Bits			
				V	N	Z	C
ADD[.W];	ADD.B	src,dst	src + dst -> dst	*	*	*	*
ADDC[.W];	ADDC.B	src,dst	src + dst + C -> dst	*	*	*	*
AND[.W];	AND.B	src,dst	src .and. dst -> dst	0	*	*	*
BIC[.W];	BIC.B	src,dst	.not.src .and. dst -> dst	-	-	-	-
BIS[.W];	BIS.B	src,dst	src .or. dst -> dst	-	-	-	-
BIT[.W];	BIT.B	src,dst	src .and. dst	0	*	*	*
CMP[.W];	CMP.B	src,dst	dst - src	*	*	*	*
DADD[.W];	DADD.B	src,dst	src + dst + C -> dst (dec)	*	*	*	*
MOV[.W];	MOV.B	src,dst	src -> dst	-	-	-	-
SUB[.W];	SUB.B	src,dst	dst + .not.src + 1 -> dst	*	*	*	*
SUBC[.W];	SUBC.B	src,dst	dst + .not.src + C -> dst	*	*	*	*
XOR[.W];	XOR.B	src,dst	src .xor. dst -> dst	*	*	*	*

Note: Operations using Status Register SR for destination

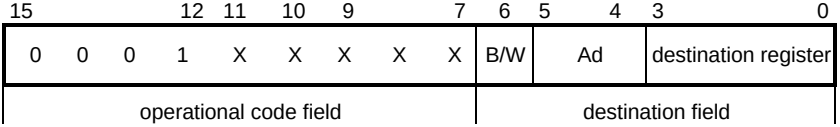
All operations using Status Register SR for destination overwrite the contents of SR with the result of that operation: the status bits are not affected as described in that operation.

Example: ADD #3,SR ; Operation: (SR) + 3 --> SR

Single operand instructions (core instructions)

The instruction format using a single operand consists of two main fields, in total 16bit:
- operational code field, 9bit with 4MSB equal '1h'
- byte operation identifier, 1bit [BW]
- destination field, 6bit [destination register + Ad]

The destination field is composed of two addressing bits and the 4bit register number (0....15). The bit position of the destination field is located in the same position as the two operand instructions. The byte identifier B/W indicates whether the instruction is executed as a byte (B/W=1) or as a word instruction (B/W=0)



				Status Bits			
				V	N	Z	C
RRA[.W];	RRA.B	dst	MSB → MSB → ...LSB → C	0	*	*	*
RRC[.W];	RRC.B	dst	C → MSB → .....LSB → C	*	*	*	*
PUSH[.W];	PUSH.B	dst	SP - 2 → SP, src → @SP	-	-	-	-
SWPB		dst	swap bytes	-	-	-	-
CALL		dst	PC+2 → @SP, dst → PC	-	-	-	-
RETI			TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT		dst	Bit7 -> Bit8 ..... Bit15	0	*	*	*

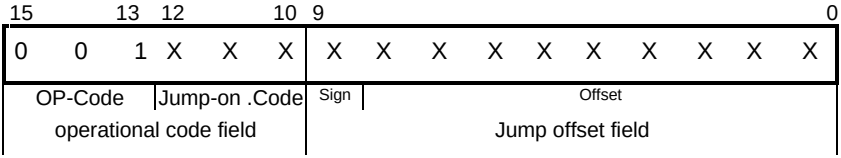
B

Conditional and unconditional Jumps (core instructions)

The instruction format for (un-)conditional jumps consists of two main fields, in total 16bit :

- operational code (OP-Code) field, 6bit
- jump offset field, 10bit

The operational code field is composed of OP-Code (3bits), and 3 bits according to the following conditions.



The conditional jumps allow jumps to addresses in the range -511 to +512 words relative to the current address. The assembler computes the signed offsets and inserts them into the opcode.

JC/JHS	Label	Jump to Label if Carry-bit is set
JEQ/JZ	Label	Jump to Label if Zero-bit is set
JGE	Label	Jump to Label if (N .XOR. V) = 0
JL	Label	Jump to Label if (N .XOR. V) = 1
JMP	Label	Jump to Label unconditionally
JN	Label	Jump to Label if Negative-bit is set
JNC/JLO	Label	Jump to Label if Carry-bit is reset
JNE/JNZ	Label	Jump to Label if Zero-bit is reset

Note: Conditional and unconditional Jumps

The conditional and unconditional Jumps do not effect the status bits.

A Jump which has been taken alters the PC with the offset:

PCnew=PCold + 2 + 2*offset.

A Jump which has not been taken continues the program with the ascending instruction.

Emulation of instructions without ROM penalty

The following instructions can be emulated with the reduced instruction set, without additional ROM words. The assembler accepts the mnemonic of the emulated instruction, and inserts the opcode of the suitable core instruction.

Note: Emulation of the following instructions

The emulation of the following instructions is possible using the contents of R2 and R3:
The register R2(CG1) contains the immediate values 2 and 4; the register R3(CG2) contains -1 or 0FFFFh, 0, +1 and +2 depending on the addressing bits As. The assembler sets the addressing bits according to the immediate value used.

Short form of emulated instructions

Mnemonic		Description	Statusbits V	N	Z	C	Emulation	
Arithmetical instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-Decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-Decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#0FFFFh,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data instructions (common use)								
CLR[.W]		Clear destination	-	-	-	-	MOV	#0,dst
CLR.B		Clear destination	-	-	-	-	MOV.B	#0,dst
CLRC		Clear carry bit	-	-	-	0	BIC	#1,SR
CLRN		Clear negative bit	-	0	-	-	BIC	#4,SR
CLRZ		Clear zero bit	-	-	0	-	BIC	#2,SR
POP	dst	Item from stack	-	-	-	-	MOV	@SP+,dst
SETC		Set carry bit	-	-	-	1	BIS	#1,SR
SETN		Set negative bit	-	1	-	-	BIS	#4,SR
SETZ		Set zero bit	-	-	1	-	BIS	#2,SR
TST[.W]	dst	Test destination	0	*	*	1	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	1	CMP.B	#0,dst
Program flow instructions								
BR	dst	Branch to	-	-	-	-	MOV	dst,PC
DINT		Disable interrupt	-	-	-	-	BIC	#8,SR
EINT		Enable interrupt	-	-	-	-	BIS	#8,SR
NOP		No operation	-	-	-	-	MOV	#0h,#0h
RET		Return from subroutine	-	-	-	-	MOV	@SP+,PC

Instruction set description - alphabetical order

This section catalogues and describes all core and emulated instructions. Some examples are given for explanation and as application hints.
The suffix .W or no suffix in the instruction mnemonic will result in a word operation.
The suffix .B at the instruction mnemonic will result in a byte operation.

*** ADC[.W]** Add carry to destination
*** ADC.B** Add carry to destination

Syntax
ADC dst or ADC.W dst
ADC.B dst

Operation
dst + C -> dst

Emulation
```
ADDC    #0,dst
ADDC.B  #0,dst
```

Description The carry C is added to the destination operand. The previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
Set if dst was incremented from 0FFh to 00, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)     ; Add LSDs
ADC     2(R12)                  ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)     ; Add LSDs
ADC.B   1(R12)          ; Add carry to MSD
```

B

ADD[.W] Add source to destination
ADD.B Add source to destination

Syntax
ADD src,dst or ADD.W src,dst
ADD.B src,dst

Operation
src + dst -> dst

Description
The source operand is added to the destination operand. The source operand is not affected, the previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the result, cleared if not.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits
OscOff, **CPUOff** and **GIE** are not affected

Example
R5 is increased by 10. The 'Jump' to TONI is performed on a carry

```
ADD     #10,R5
JC      TONI        ; Carry occurred
.....               ; No carry
```

Example
R5 is increased by 10. The 'Jump' to TONI is performed on a carry

```
ADD.B   #10,R5      ; Add 10 to Lowbyte of R5
JC      TONI        ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
.....               ; No carry
```

B

ADDC[.W] Add source and carry to destination.
ADDC.B Add source and carry to destination.

Syntax ADDC src,dst or ADDC.W src,dst
ADDC.B src,dst

Operation src + dst + C -> dst

Description The source operand and the carry C are added to the destination operand. The source operand is not affected, the previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 32-bit counter pointed to by R13 is added to a 32-bit counter eleven **words** (20/2 + 2/2) above pointer in R13.

```
ADD     @R13+,20(R13)     ; ADD LSDs with no carryin
ADDC    @R13+,20(R13)     ; ADD MSDs with carry
...                       ; resulting from the LSDs
```

Example The 24-bit counter pointed to by R13 is added to a 24-bit counter eleven words above pointer in R13.

```
ADD.B   @R13+,10(R13)     ; ADD LSDs with no carryin
ADDC.B  @R13+,10(R13)     ; ADD medium Bits with carry
ADDC.B  @R13+,10(R13)     ; ADD MSDs with carry
...                       ; resulting from the LSDs
```

AND[.W] source AND destination
AND.B source AND destination

Syntax

```
AND     src,dst    or      AND.W   src,dst
AND.B   src,dst
```

Operation src .AND. dst -> dst

Description The source operand and the destination operand are logically AND'ed. The result is placed into the destination.

Status Bits
N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI

```
MOV     #0AA55h,R5      ; Load mask into register R5
AND     R5,TOM          ; mask word addressed by TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
;
;
;
;       or
;
;
;
AND     #0AA55h,TOM
JZ      TONI
```

Example The bits of mask #0A5h are logically AND'ed with the Lowbyte TOM. If the result is zero, a branch is taken to label TONI

```
AND.B   #0A5h,TOM       ; mask Lowbyte TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
```

B

BIC[.W] Clear bits in destination
BIC.B Clear bits in destination

Syntax

BIC src,dst or BIC.W src,dst
BIC.B src,dst

Operation

.NOT.src .AND. dst -> dst

Description

The inverted source operand and the destination operand are logically AND'ed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits

OscOff, **CPUOff** and **GIE** are not affected

Example

The 6 MSBs of the RAM word LEO are cleared.

```
BIC#0FC00h,LEO          ; Clear 6 MSBs in MEM(LEO)
```

Example

The 5 MSBs of the RAM byte LEO are cleared.

```
BIC.B   #0F8h,LEO       ; Clear 5 MSBs in Ram location LEO
```

Example

The Portpins P0 and P1 are cleared.

```
P0OUT   .equ    011h            ;Definition of the Portaddress
P0_0    .equ    01h
P0_1    .equ    02h

BIC.B   #P0_0+P0_1,&P0OUT  ;Set P0.0 and P0.1 to low
```

B

BIS[.W] Set bits in destination
BIS.B Set bits in destination

Syntax

BIS src,dst or BIS.W src,dst
BIS.B src,dst

Operation

src .OR. dst -> dst

Description

The source operand and the destination operand are logically OR'ed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits

OscOff, **CPUOff** and **GIE** are not affected

Example

The 6 LSB's of the RAM word TOM are set.

```
BIS     #003Fh,TOM  ;  set the 6 LSB's in RAM location TOM
```

Example

Start an A/D-conversion

```
ASOC    .equ    1       ; Start of Conversion bit
ACTL    .equ    114h    ; ADC-Control Register

BIS     #ASOC,&ACTL     ; Start A/D-conversion
```

Example

The 3 MSBs of the RAM byte TOM are set.

```
BIS.B   #0E0h,TOM       ; set the 3 MSBs in RAM location TOM
```

Example

The Portpins P0 and P1 are set to high

```
P0OUT   .equ    011h
P0      .equ    01h
P1      .equ    02h

BIS.B   #P0+P1,&P0OUT
```

B

BIT[.W] Test bits in destination
BIT.B Test bits in destination

Syntax BIT src,dst or BIT.W src,dst

Operation src .AND. dst

Description The source operand and the destination operand are logically AND'ed. The result affects only the Status Bits. The source and destination operands are not affected.

Status Bits
N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example If bit 9 of R8 is set, a branch is taken to label TOM.

```
BIT     #0200h,R8       ; bit 9 of R8 set ?
JNZ     TOM             ; Yes, branch to TOM
...                     ; No, proceed
```

Example Determine which A/D-Channel is configured by the MUX

```
ACTL    .equ    114h    ; ADC Control Register

BIT     #4,&ACTL        ; Is Channel 0 selected ?
jnz     END             ; Yes, branch to END
```

Example If bit 3 of R8 is set, a branch is taken to label TOM.

```
BIT.B   #8,R8
JC      TOM
```

BIT (continued)

Example The receive bit RCV of a serial communication is tested. Since while using the BIT instruction to test a single bit the carry is equal to the state of the tested bit, the carry is ; used by the subsequent instruction: the read info is shifted into the register RECBUF.

```
;
; Serial communication with LSB is shifted first:
                                        ; xxxx    xxxx    xxxx    xxxx
        BIT.B   #RCV,RCCTL              ; Bit info into carry
        RRC     RECBUF                  ; Carry -> MSB of RECBUF
                                        ; cxxx  xxxx
        ......                          ; repeat previous two instructions
        ......                          ; 8 times
                                        ; cccc  cccc
                                        ; ^       ^
                                        ; MSB     LSB

; Serial communication with MSB is shifted first:
        BIT.B   #RCV,RCCTL              ; Bit info into carry
        RLC.B   RECBUF                  ; Carry -> LSB of RECBUF
                                        ; xxxx    xxxc
        ......                          ; repeat previous two instructions
        ......                          ; 8 times
                                        ; cccc    cccc
                                        ; |       LSB
                                        ; MSB
```

B

* BR, BRANCH

Branch to ..... destination

Syntax BR dst

Operation dst -> PC

Emulation MOV dst,PC

Description An unconditional branch is taken to an address anywhere in the 64 K address space. All source addressing modes may be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected

Examples Examples for all addressing modes are given

```
BR    #EXEC    ;Branch to label EXEC or direct branch (e.g. #0A4h)
               ; Core instruction  MOV  @PC+,PC

BR    EXEC     ; Branch to the address contained in EXEC
               ; Core instruction  MOV  X(PC),PC
               ; Indirect address

BR    &EXEC    ; Branch to the address contained in absolute
               ; address EXEC
               ; Core instruction  MOV  X(0),PC
               ; Indirect address

BR    R5       ; Branch to the address contained in R5
               ; Core instruction  MOV  R5,PC
               ; Indirect R5

BR    @R5      ; Branch to the address contained in the word R5
               ; points to.
               ; Core instruction  MOV  @R5,PC
               ; Indirect, indirect R5

BR    @R5+     ; Branch to the address contained in the word R5
               ; points to and increments pointer in R5 afterwards.
               ; The next time - S/W flow uses R5 pointer - it can
               ; alter the program execution due to access to
               ; next address in a table, pointed by R5
               ; Core instruction  MOV  @R5,PC
               ; Indirect, indirect R5 with autoincrement

BR    X(R5)    ; Branch to the address contained in the address
               ; pointed to  by R5 + X  (e.g. table with address
               ; starting at X). X can be an address or a label
               ; Core instruction MOV  X(R5),PC
               ; Indirect indirect R5 + X
```

CALL Subroutine

Syntax CALL dst

Operation

dst	-> tmp	dst is evaluated and stored
SP - 2	-> SP	
PC	-> @SP	updated PC to TOS
tmp	-> PC	saved dst to PC

Description A subroutine call is made to an address anywhere in the 64-K-address space. All addressing modes may be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.

Status Bits Status bits are not affected

Example Examples for all addressing modes are given

```
CALL   #EXEC   ; Call on label EXEC or immediate address (e.g.
               ; #0A4h)
               ; SP-2 → SP,  PC+2 → @SP,  @PC+ → PC
CALL   EXEC    ; Call on the address contained in EXEC
               ; SP-2 → SP,  PC+2 → @SP,  X(PC) → PC
               ; Indirect address
CALL   &EXEC   ; Call on the address contained in absolute address
               ; EXEC
               ; SP-2 → SP,  PC+2 → @SP,  X(PC) → PC
               ; Indirect address
CALL   R5      ; Call on the address contained in R5
               ; SP-2 → SP,  PC+2 → @SP,  R5 → PC
               ; Indirect R5
CALL   @R5     ; Call on the address contained in the word R5
               ; points
               ; to
               ; SP-2 → SP,  PC+2 → @SP,  @R5 → PC
               ; Indirect, indirect R5
CALL   @R5+    ; Call on the address contained in the word R5 points
               ; to and increments pointer in R5. The next time -
               ; S/W flow uses R5 pointer - it can alter the
               ; program execution due to access to next address
               ; in a table, pointed ; to by R5
               ; SP-2 → SP,  PC+2 → @SP,  @R5 → PC
               ; Indirect, indirect R5 with autoincrement
CALL   X(R5)   ; Call on the address contained in the address pointed
               ; to by R5 + X  (e.g. table with address starting at X)
               ; X can be an address or a label
               ; SP-2 → SP,  PC+2 → @SP,  X(R5) → PC
               ; Indirect indirect R5 + X
```

*** CLR[.W]** Clear destination
*** CLR.B** Clear destination

Syntax
CLR dst or CLR.W dst
CLR.B dst

Operation
0 -> dst

Emulation
MOV #0,dst
MOV.B #0,dst

Description The destination operand is cleared.

Status Bits Status bits are not affected

Example RAM word TONI is cleared

```
CLR     TONI    ; 0 -> TONI
```

Example Register R5 is cleared

```
CLR     R5
```

Example RAM byte TONI is cleared

```
CLR.B   TONI    ; 0 -> TONI
```

*** CLRC** Clear carry bit

Syntax CLRC

Operation 0 -> C

Emulation BIC #1,SR

Description The Carry Bit C is cleared. The clear carry instruction is a word instruction.

Status Bits
N: Not affected
Z: Not affected
C: Cleared
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16bit decimal counter pointed to by R13 is added to a 32bit counter pointed to by R12.

```
CLRC                    ; C=0: Defines start
DADD    @R13,0(R12)     ; add 16bit counter to Lowword of 32bit
                        ; counter
DADC    2(R12)          ; add carry to Highword of 32bit counter
```

* CLRN Clear Negative bit

Syntax CLRN

Operation 0 → N
or
(.NOT.src .AND. dst -> dst)

Emulation BIC #4,SR

Description The constant 04h is inverted (0FFFBh) and the destination operand are logically AND'ed. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits
N: Reset to 0
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The Negative bit in the status register is cleared. This avoids the special treatment of the called subroutine with negative numbers.

```
        CLRN
        CALL    SUBR
        ......
        ......
SUBR    JN      SUBRET      ; If input is negative: do nothing and return
        ......
        ......
        ......
SUBRET  RET
```

*** CLRZ** Clear Zero bit

Syntax CLRZ

Operation 0 → Z
or
(.NOT.src .AND. dst -> dst)

Emulation BIC #2,SR

Description The constant 02h is inverted (0FFFDh) and the destination operand are logically AND'ed. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits
N: Not affected
Z: Reset to 0
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The Zero bit in the status register is cleared.

```
CLRZ
```

CMP[.W] compare source and destination
CMP.B compare source and destination

Syntax
CMP src,dst or CMP.W src,dst
CMP.B src,dst

Operation
dst + .NOT.src + 1
or
(dst - src)

Description The source operand is subtracted from the destination operand. This is made by adding of the 1's complement of the source operand plus 1. The two operands are not affected and, the result is not stored; only the status bits are affected.

Status Bits
N: Set if result is negative, reset if positive (src >= dst)
Z: Set if result is zero, reset otherwise (src = dst)
C: Set if there is a carry from the MSB of the result, reset if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R5 and R6 are compared. If they are equal, the program continues at the label EQUAL

```
CMP    R5,R6      ; R5 = R6 ?
JEQ    EQUAL      ; YES, JUMP
```

Example Two RAM blocks are compared. If they not equal, the program branches to the label ERROR

```
       MOV    #NUM,R5              ;number of words to be compared
L$1    CMP    &BLOCK1,&BLOCK2      ;Are Words equal ?
       JNZ    ERROR                ;No, branch to ERROR
       DEC    R5                   ;Are all words compared?
       JNZ    L$1                  ;No, another compare
```

Example The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL

```
       CMP.B EDE,TONI              ; MEM(EDE) = MEM(TONI) ?
       JEQ   EQUAL                 ; YES, JUMP
```

B

CMP.B (continued)

Example Check two Keys, which are connected to the Portpin P0 and P1. If key1 is pressed, the program branches to the label MENU1; if key2 is pressed, the program branches to MENU2.

```
P0IN    .EQU    010h
KEY1    .EQU    01h
KEY2    .EQU    02h

        CMP.B   #KEY1,&P0IN
        JEQ     MENU1
        CMP.B   #KEY2,&P0IN
        JEQ     MENU2
```

*** DADC[.W]** Add carry decimally
*** DADC.B** Add carry decimally

Syntax
DADC dst o DADC.W src,dst
DADC.B dst

Operation
dst + C -> dst (decimally)

Emulation
DADD #0,dst
DADD.B #0,dst

Description The Carry Bit C is added decimally to the destination

Status Bits
N: Set if MSB is 1
Z: Set if dst is 0, reset otherwise
C: Set if destination increments from 9999 to 0000, reset otherwise
Set if destination increments from 99 to 00, reset otherwise
V: Undefined

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 4-digit decimal number contained in R5 is added to an 8-digit decimal number pointed to by R8

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD  R5,0(R8)      ; Add LSDs + C
DADC  2(R8)         ; Add carry to MSD
```

Example The 2-digit decimal number contained in R5 is added to an 4-digit decimal number pointed to by R8

```
CLRC                ; Reset carry
                    ; next instruction's start condition is
                    ; defined
DADD.B  R5,0(R8)    ; Add LSDs + C
DADC    1(R8)       ; Add carry to MSDs
```

B

DADD[.W] source and carry added decimally to destination
DADD.B source and carry added decimally to destination

Syntax DADD src,dst or DADD.W src,dst
DADD.B src,dst

Operation src + dst + C -> dst (decimally)

Description The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry C are added decimally to the destination operand. The source operand is not affected, the previous contents of the destination are lost. The result is not defined for non-BCD numbers.

Status Bits
N: Set if the MSB is 1, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if the result is greater than 9999.
Set if the result is greater than 99.
V: Undefined

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 8-digit-BCD-number contained in R5 and R6 is added decimally to a 8-digit-BCD-number contained in R3 and R4 (R6 and R4 contain the MSDs).

```
CLRC                ; CLEAR CARRY
DADD    R5,R3       ; add LSDs
DADD    R6,R4       ; add MSDs with carry
JC      OVERFLOW    ; If carry occurs go to error handling routine
```

Example The 2-digit decimal counter in RAMbyte CNT is incremented by one.

```
CLRC                ; clear Carry
DADD.B  #1,CNT      ; increment decimal counter
```

or

```
SETC
DADD.B  #0,CNT      ; ≡ DADC.B  CNT
```

B

* DEC[.W] Decrement destination
* DEC.B Decrement destination

Syntax
DEC dst or DEC.W dst
DEC.B dst

Operation
dst - 1 -> dst

Emulation
SUB #1,dst

Emulation
SUB.B #1,dst

Description
The destination operand is decremented by one. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 1, reset otherwise
C: Reset if dst contained 0, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08000h, otherwise reset.
Set if initial value of destination was 080h, otherwise reset.

Mode Bits
OscOff, CPUOff and **GIE** are not affected

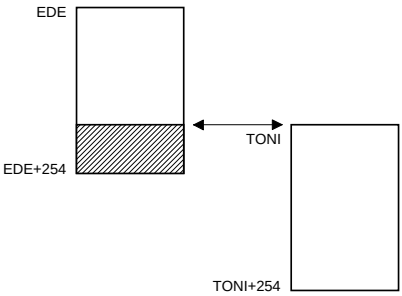
* DEC (continued)

Example R10 is decremented by 1

```
        DEC    R10          ; Decrement R10
```

```
; Move a block of 255 bytes from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
range ; EDE to EDE+0FEh
;
        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

; Do not transfer tables with the routine above with this overlap:



Example Memory byte at address LEO is decremented by 1

```
        DEC.B   LEO     ; Decrement MEM(LEO)
```

```
; Move a block of 255 bytes from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;
        MOV     #EDE,R6
        MOV.B   #255,LEO
L$1     MOV.B   @R6+,TONI-EDE-1(R6)
        DEC.B   LEO
        JNZ     L$1
```

B

*** DECD[.W]** Double-Decrement destination
*** DECD.B** Double-Decrement destination

Syntax DECD dst or DECD.W dst
DECD.B dst

Operation dst - 2 -> dst

Emulation SUB #2,dst
Emulation SUB.B #2,dst

Description The destination operand is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 2, reset otherwise
C: Reset if dst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08001 or 08000h, otherwise reset.
Set if initial value of destination was 081 or 080h, otherwise reset.

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R10 is decremented by 2

```
        DECD    R10             ; Decrement R10 by two
```

```
; Move a block of 255 words from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;
        MOV     #EDE,R6
        MOV     #510,R10
L$1     MOV     @R6+,TONI-EDE-2(R6)
        DECD    R10
        JNZ     L$1
```

Example Memory at location LEO is decremented by 2

```
        DECD.B  LEO             ; Decrement MEM(LEO)
```

Decrement status byte STATUS by 2

```
        DECD.B  STATUS
```

*** DINT** Disable (general) interrupts

Syntax DINT

Operation 0 → GIE
or
(0FFF7h .AND. SR → SR / .NOT.src .AND. dst -> dst)

Emulation BIC #8,SR

Description All interrupts are disabled.
The constant #08h is inverted and logically AND'ed with the status register SR. The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **GIE** is reset.
OscOff and **CPUOff** are not affected

Example The general interrupt enable bit GIE in the status register is cleared to allow a non disrupted move of a 32bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                    ; All interrupt events using the GIE bit are
                        ; disabled
MOV   COUNTHI,R5        ; Copy counter
MOV   COUNTLO,R6
EINT                    ; All interrupt events using the GIE bit are
                        ; enabled
```

Note: Disable Interrupt

The instruction following the disable interrupt instruction DINT is executed when the interrupt request becomes active during execution of DINT. If any code sequence needs to be protected from being interrupted, the DINT instruction should be executed at least one instruction before this sequence.

*** EINT** Enable (general) interrupts

Syntax EINT

Operation 1 → GIE
or
(0008h .OR. SR -> SR / .NOT.src .OR. dst -> dst)

Emulation BIS #8,SR

Description All interrupts are enabled.
The constant #08h and the status register SR are logically OR'ed. The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **GIE** is set.
OscOff and **CPUOff** are not affected

Example The general interrupt enable bit GIE in the status register is set.

```
; Interrupt routine of port P0.2 to P0.7
; The interrupt level is the lowest in the system
; P0IN is the address of the register where all port bits are read. P0IFG is the address of
; the register where all interrupt events are latched.
;
            PUSH.B   &P0IN
            BIC.B    @SP,&P0IFG ; Reset only accepted flags
            EINT                ; Preset port 0 interrupt flags stored on stack
                                ; other interrupts are allowed
            BIT      #Mask,@SP
            JEQ      MaskOK     ; Flags are present identically to mask: Jump
            ......
MaskOK      BIC      #Mask,@SP
            ......
            INCD     SP         ; Housekeeping: Inverse to PUSH instruction
                                ; at the start of interrupt subroutine. Corrects
                                ; the  stack pointer.
            RETI
```

B

Note: Enable Interrupt

The instruction following the enable interrupt instruction EINT is executed anyway, even if an interrupt service request is pending.

*** INC[.W]** Increment destination
*** INC.B** Increment destination

Syntax
INC dst or INC.W dst
INC.B dst

Operation
dst + 1 -> dst

Emulation
ADD #1,dst

Description
The destination operand is incremented by one. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits
OscOff, **CPUOff** and **GIE** are not affected

Example
The item on the top of a software stack (not the system stack) for byte data is removed.

```
SSP  .EQU   R4
;
     INC    SSP  ; Remove TOSS (top of SW stack) by increment
                 ; Do not use INC.B since SSP is a word register
```

Example
The status byte of a process STATUS is incremented. When it is equal to eleven, a branch to OVFL is taken.

```
INC.B   STATUS
CMP.B   #11,STATUS
JEQ     OVFL
```

*** INCD[.W]** Double-Increment destination
*** INCD.B** Double-Increment destination

Syntax INCD dst or INCD.W dst
INCD.B dst

Operation dst + 2 -> dst

Emulation ADD #2,dst
Emulation ADD.B #2,dst

Description The destination operand is incremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFEh, reset otherwise
Set if dst contained 0FEh, reset otherwise
C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
Set if dst contained 0FEh or 0FFh, reset otherwise
V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The item on the top of the stack is removed without the use of a register.

```
.......
PUSH    R5    ; R5 is the result of a calculation, which is stored
              ; in the system stack
INCD    SP    ; Remove TOS by double-increment from stack
              ; Do not use INCD.B, SP is a word aligned
              ; register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B  0(SP)   ; Byte on TOS is increment by two
```

B

*** INV[.W]** Invert destination
*** INV.B** Invert destination

Syntax
```
INV     dst
INV.B   dst
```

Operation .NOT.dst -> dst

Emulation
```
XOR     #0FFFFh,dst
```
Emulation
```
XOR.B   #0FFh,dst
```

Description The destination operand is inverted. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if initial destination operand was negative, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Content of R5 is negated (two's complement).

```
MOV   #00Aeh,R5     ;                       R5 = 000AEh
INV   R5            ; Invert R5,            R5 = 0FF51h
INC   R5            ; R5 is now negated,    R5 = 0FF52h
```

Example Content of memory byte LEO is negated.

```
MOV.B   #0AEh,LEO   ;                       MEM(LEO) = 0AEh
INV.B   LEO         ; Invert LEO,           MEM(LEO) = 051h
INC.B   LEO         ; MEM(LEO) is negated,  MEM(LEO) = 052h
```

B

JC Jump if carry set
JHS Jump if higher or same

Syntax

```
JC      label
JHS     label
```

Operation

if C = 1: PC + 2*offset -> PC
if C = 0: execute following instruction

Description The Carry Bit C of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits Status bits are not affected

Example The signal of input P0IN.1 is used to define or control the program flow.

```
BIT     #10h,&P0IN  ; State of signal -> Carry
JC      PROGA       ; If carry=1 then execute program routine A
......              ; Carry=0, execute program here
```

Example R5 is compared to 15. If the content is higher or same branch to LABEL.

```
CMP     #15,R5
JHS     LABEL       ; Jump is taken if R5 ≥ 15
......              ; Continue here if R5 < 15
```

JEQ, JZ

Jump if equal, Jump if zero

Syntax JEQ label, JZ label

Operation
if Z = 1: PC + 2*offset -> PC
if Z = 0: execute following instruction

Description The Zero Bit Z of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If Z is not set, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example Jump to address TONI if R7 contains zero.

```
TST     R7          ; Test R7
JZ      TONI        ; if zero: JUMP
```

Example Jump to address LEO if R6 is equal to the table contents.

```
CMP     R6,Table(R5)  ; Compare content of R6 with content of
                      ; MEM(Table address + content of R5)
JEQ     LEO           ; Jump if both data are equal
.....                 ; No, data are not equal, continue here
```

Example Branch to LABEL if R5 is 0.

```
TST     R5
JZ      LABEL
.....
```

B

JGE

Jump if greater or equal

Syntax JGE label

Operation if (N .XOR. V) = 0 then jump to label: PC + 2*offset -> PC
if (N .XOR. V) = 1 then execute following instruction

Description The negative bit N and the overflow bit V of the Status Register are tested. If both N and V are set or reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If only one is set, the next instruction following the jump is executed.
This allows comparison of signed integers.

Status Bits Status bits are not affected

Example When the content of R6 is greater or equal the memory pointed to by R7 the program continues at label EDE.

```
CMP     @R7,R6      ; R6 ≥ (R7)?, compare on signed numbers
JGE     EDE         ; Yes, R6 ≥ (R7)
.....               ; No, proceed
.....
.....
```

JL Jump if less

Syntax JL label

Operation if (N .XOR. V) = 1 then jump to label: PC + 2*offset -> PC
if (N .XOR. V) = 0 then execute following instruction

Description The negative bit N and the overflow bit V of the Status Register are tested. If only one is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If both N and V are set or reset, the next instruction following the jump is executed.
This allows comparison of signed integers.

Status Bits Status bits are not affected

Example When the content of R6 is less than the memory pointed to by R7 the program continues at label EDE.

```
CMP     @R7,R6      ; R6 < (R7)?,  compare on signed numbers
JL      EDE         ; Yes, R6 < (R7)
.....               ; No, proceed
.....
.....
```

JMP Jump unconditionally

Syntax JMP label

Operation PC + 2*offset -> PC

Description The 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter.

Status Bits Status bits are not affected

Hint This 1word instruction replaces the BRANCH instruction in the range of -511 to +512 words, relative to the current program counter.

JN

Jump if negative

Syntax JN label

Operation if N = 1: PC + 2*offset -> PC
if N = 0: execute following instruction

Description The negative bit N of the Status Register is tested. If it is set, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If N is reset, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.

```
        SUB   R5,COUNT     ; COUNT - R5 -> COUNT
        JN    L$1          ; If negative continue with COUNT=0at PC=L$1
        ......             ; Continue with COUNT≥0
        ......
        ......
        ......
L$1     CLR   COUNT
        ......
        ......
        ......
```

JNC Jump if carry not set
JLO Jump if lower

Syntax
```
JNC     label
JNC     label
```

Operation
if C = 0: PC + 2*offset -> PC
if C = 1: execute following instruction

Description The Carry Bit C of the Status Register is tested. If it is reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits status bits are not affected

Example The result in R6 is added in BUFFER. If an overflow occurs an error handling routine at address ERROR is going to be used.

```
        ADD     R6,BUFFER       ; BUFFER + R6 -> BUFFER
        JNC     CONT            ; No carry, jump to CONT
ERROR   ......                  ; Error handler start
        ......
        ......
        ......
CONT    ......                  ; Continue with normal program flow
        ......
        ......
```

Example Branch to STL2 if byte STATUS contains 1 or 0.

```
        CMP.B   #2,STATUS
        JLO     STL2            ; STATUS < 2
        ......                  ; STATUS ≥ 2, continue here
```

JNE, JNZ

Jump if not equal, Jump if not zero

Syntax JNE label, JNZ label

Operation if Z = 0: PC + 2*offset -> PC
if Z = 1: execute following instruction

Description The Zero Bit Z of the Status Register is tested. If it is reset, the 10-bit signed offset contained in the LSB's of the instruction is added to the Program Counter. If Z is set, the next instruction following the jump is executed.

Status Bits Status bits are not affected

Example Jump to address TONI if R7 and R8 have different contents

```
CMP   R7,R8      ; COMPARE R7 WITH R8
JNE   TONI       ; if different: Jump
.....            ; if equal, continue
```

MOV[.W] Move source to destination
MOV.B Move source to destination

Syntax MOV src,dst or MOV.W src,dst
MOV.B src,dst

Operation src -> dst

Description The source operand is moved to the destination.
The source operand is not affected, the previous contents of the destination are lost.

Status Bits Status bits are not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The contents of table EDE (word data) are copied to table TOM. The length of the tables should be 020h locations.

```
        MOV   #EDE,R10                ; Prepare pointer
        MOV   #020h,R9                ; Prepare counter
Loop    MOV   @R10+,TOM-EDE-2(R10)    ; Use pointer in R10 for both tables
        DEC   R9                      ; Decrement counter
        JNZ   Loop                    ; Counter ≠ 0, continue copying
        ......                        ; Copying completed
        ......
        ......
```

Example The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations.

```
        MOV   #EDE,R10                ; Prepare pointer
        MOV   #020h,R9                ; Prepare counter
Loop    MOV.B @R10+,TOM-EDE-1(R10)    ; Use pointer in R10 for
                                      ; both tables
        DEC   R9                      ; Decrement counter
        JNZ   Loop                    ; Counter ≠ 0, continue
                                      ; copying
        ......                        ; Copying completed
        ......
        ......
```

B

*** NOP** No operation

Syntax NOP

Operation None

Emulation MOV #0,#0

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected

The NOP instruction is mainly used for two purposes:

- hold one, two or three memory words
- adjust software timing

Note: Other instructions can be used to emulate no operation

Other instructions can be used to emulate no-operation instruction, using different numbers of cycles and different numbers of code words.

Examples:

```
MOV   0(R4),0(R4)     ; 6 cycles, 3 words
MOV   @R4,0(R4)       ; 5 cycles, 2 words
BIC   #0,EDE(R4)      ; 4 cycles, 2 words
JMP   $+2             ; 2 cycles, 1 word
BIC   #0,R5           ; 1 cycles, 1 word.
```

B

*** POP[.W]** Pop word from stack to destination
*** POP.B** Pop byte from stack to destination

Syntax
```
POP     dst
POP.B   dst
```

Operation
```
@SP   -> dst
SP + 2 -> SP
```

Emulation
```
MOV   @SP+,dst    or    MOV.W   @SP+,dst
MOV.B   @SP+,dst
```

Description The stack location pointed to by the Stack Pointer (TOS) is moved to the destination. The Stack Pointer is incremented by two afterwards.

Status Bits Status bits are not affected

Example The contents of R7 and the Status Register are restored from the stack.

```
POP    R7          ; Restore R7
POP    SR          ; Restore status register
```

Example The content of RAM byte LEO is restored from the stack.

```
POP.B    LEO       ; The Low byte of the stack is moved to LEO.
```

Example The content of R7 is restored from the stack.

```
POP.B    R7        ; The Low byte of the stack is moved to R7,
                   ; the High byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the Status Register are restored from the stack.

```
POP.B    0(R7)     ; The Low byte of the stack is moved to the
                   ; the byte which is pointed to by R7
                   : Example:  R7 = 203h
                   ;           Mem(R7) = Low Byte of system stack
                   : Example:   R7 = 20Ah
                   ;           Mem(R7) = Low Byte of system stack
POP      SR
```

Note: The system Stack Pointer

The system Stack Pointer SP is always incremented by two, independent of the byte suffix. This is mandatory since the system Stack Pointer is used not only by POP instructions; it is also used by the RETI instruction.

PUSH[.W] Push word onto stack
PUSH.B Push byte onto stack

Syntax PUSH src or PUSH.W src
PUSH.B src

Operation SP - 2 → SP
src → @SP

Description The Stack Pointer is decremented by two, then the source operand is moved to the RAM word addressed by the Stack Pointer (TOS).

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The contents of the Status Register and R8 are saved on the stack.

```
PUSH    SR      ; save status register
PUSH    R8      ; save R8
```

Example The content of the peripheral TCDAT is saved on the stack.

```
PUSH.B  &TCDAT  ; save data from 8bit peripheral module,
                ; address TCDAT, onto stack
```

Note: The system Stack Pointer

The system Stack Pointer SP is always decremented by two, independent of the byte suffix. This is mandatory since the system Stack Pointer is used not only by PUSH instruction; it is also used by the interrupt routine service.

*** RET** Return from subroutine

Syntax RET

Operation @SP→ PC
SP + 2 → SP

Emulation MOV @SP+,PC

Description The return address pushed onto the stack by a CALL instruction is moved to the Program Counter. The program continues at the code address following the subroutine call.

Status Bits Status bits are not affected

RETI Return from Interrupt

Syntax RETI

Operation

TOS → SR
SP + 2 → SP
TOS → PC
SP + 2 → SP

Description

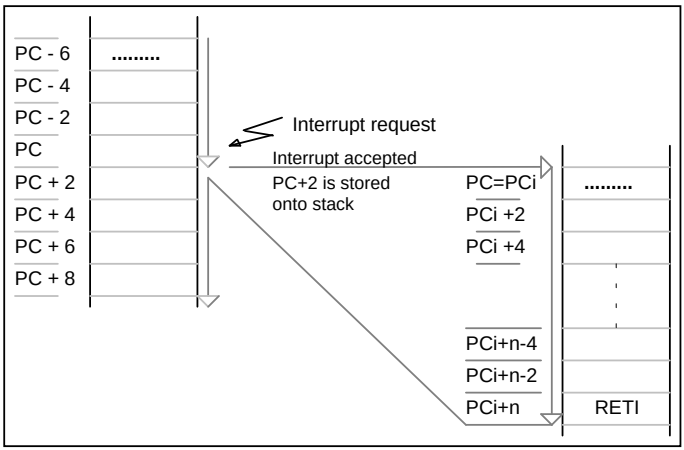
1. The status register is restored to the value at the beginning of the interrupt service routine. This is performed by replacing the present contents of SR with the contents of TOS memory. The stack pointer SP is incremented by two.
2. The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restore is performed by replacing present contents of PC with the contents of TOS memory. The stack pointer SP is incremented.

Status Bits

N: restored from system stack
Z: restored from system stack
C: restored from system stack
V: restored from system stack

Mode Bits **OscOff**, **CPUOff** and **GIE** are restored from system stack

Example Main program is interrupted



B

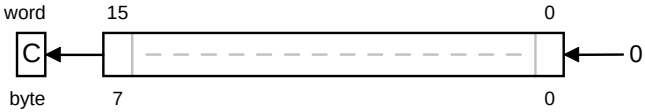
*** RLA[.W]** Rotate left arithmetically
*** RLA.B** Rotate left arithmetically

Syntax
RLA dst or RLA.W dst
RLA.B dst

Operation
C <- MSB <- MSB-1 .... LSB+1 <- LSB <- 0

Emulation
ADD dst,dst
ADD.B dst,dst

Description
The destination operand is shifted left one position. The MSB is shifted into the carry C, the LSB is filled with 0. The RLA instruction acts as a signed multiplication with 2.
An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed: the result has changed sign.



An overflow occurs if dst ≥ 040h and dst < 0C0h before operation is performed: the result has changed sign.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs -
the initial value is 04000h ≤ dst < 0C000h; otherwise it is reset
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 0C0h; otherwise it is reset

Mode Bits
OscOff, **CPUOff** and **GIE** are not affected

* RLA (continued)

Example R7 is multiplied by 4.

```
RLA     R7      ; Shift left R7  (x 2) - emulated by   ADD  R7,R7
RLA     R7      ; Shift left R7  (x 4) - emulated by   ADD  R7,R7
```

Example Lowbyte of R7 is multiplied by 4.

```
RLA.B   R7      ; Shift left Lowbyte of R7  (x 2) - emulated by
                ; ADD.B  R7,R7
RLA.B   R7      ; Shift left Lowbyte of R7  (x 4) - emulated by
                ; ADD.B  R7,R7
```

Note: RLA substitution

The Assembler does not recognize the instruction
RLA @R5+ nor RLA.B @R5+.
It must be substituted by
ADD @R5+,-2(R5) or ADD.B @R5+,-1(R5).

B

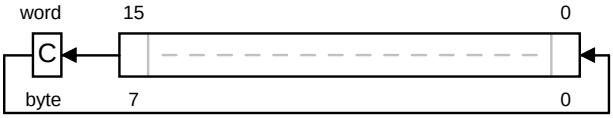
*** RLC[.W]** Rotate left through carry
*** RLC.B** Rotate left through carry

Syntax RLC dst or RLC.W dst
RLC.B dst

Operation C <- MSB <- MSB-1 .... LSB+1 <- LSB <- C

Emulation ADDC dst,dst

Description The destination operand is shifted left one position. The carry C is shifted into the LSB, the MSB is shifted into the carry C.



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if arithmetic overflow occurs otherwise reset
Set if 03FFFh < $dst_{initial}$ < 0C000h, otherwise reset
Set if 03Fh < $dst_{initial}$ < 0C0h, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

* RLC (continued)

Example R5 is shifted left one position.

```
RLC     R5          ; (R5 x 2) + C -> R5
```

Example The information of input P0IN.1 is to be shifted into LSB of R5.

```
BIT.B   #2,&P0IN    ; Information -> Carry
RLC     R5          ; Carry=P0in.1 -> LSB of R5
```

Example Content of MEM(LEO) is shifted left one position.

```
RLC.B   LEO         ; Mem(LEO) x 2 + C -> Mem(LEO)
```

Example The information of input P0IN.1 is to be shifted into LSB of R5.

```
BIT.B   #2,&P0IN    ; Information -> Carry
RLC.B   R5          ; Carry=P0in.1 -> LSB of R5
                    ; High byte of R5 is reset
```

Note: RLC and RLC.B emulation

The Assembler does not recognize the instruction

```
RLC     @R5+.
```

It must be substituted by

```
ADDC    @R5+,-2(R5).
```

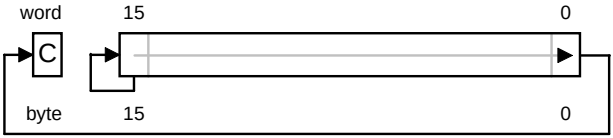
B

RRA[.W] Rotate right arithmetically
RRA.B Rotate right arithmetically

Syntax
RRA dst or RRA.W dst
RRA.B dst

Operation
MSB -> MSB, MSB -> MSB-1, ... LSB+1 -> LSB, LSB -> C

Description
The destination operand is shifted right one position. The MSB is shifted into the MSB, the MSB is shifted into the MSB-1, the LSB+1 is shifted into the LSB.



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits
OscOff, **CPUOff** and **GIE** are not affected

RRA (continued)

Example R5 is shifted right one position. The MSB remains with the old value. It operates equal to an arithmetic division by 2.

```
                RRA     R5          ; R5/2 -> R5

;               The value in R5 is multiplied by 0.75 (0.5 + 0.25)
;
                PUSH    R5          ; hold R5 temporarily using stack
                RRA     R5          ; R5 x 0.5  ->  R5
                ADD     @SP+,R5     ; R5 x 0.5 + R5 = 1.5 x R5  -> R5
                RRA     R5          ; (1.5 x R5) x 0.5 = 0.75 x R5  -> R5
                ......
                ......
; OR
;
                RRA     R5          ; R5 x 0.5  ->  R5
                PUSH    R5          ; R5 x 0.5  ->  TOS
                RRA     @SP         ; TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5  -> TOS
                ADD     @SP+,R5     ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5  -> R5
                ......
```

Example The Lowbyte of R5 is shifted right one position. The MSB remains with the old value. It operates equal to an arithmetic division by 2.

```
                RRA.B   R5          ; R5/2 -> R5: Operation is on Low byte only
                                    ; High byte of R5 is reset

;               The value in R5 - Low byte only! - is multiplied by 0.75 (0.5 + 0.25)
;
                PUSH.B  R5          ; hold Low byte of R5 temporarily using stack
                RRA.B   R5          ; R5 x 0.5  ->  R5
                ADD.B   @SP+,R5     ; R5 x 0.5 + R5 = 1.5 x R5  -> R5
                RRA.B   R5          ; (1.5 x R5) x 0.5 = 0.75 x R5  -> R5
                ......
; OR
;
                RRA.B   R5          ; R5 x 0.5  ->  R5
                PUSH.B  R5          ; R5 x 0.5  ->  TOS
                RRA.B   @SP         ;TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25x R5  -> TOS
                ADD.B   @SP+,R5     ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5  -> R5
                ......
```

B

RRC[.W] Rotate right through carry
RRC.B Rotate right through carry

Syntax

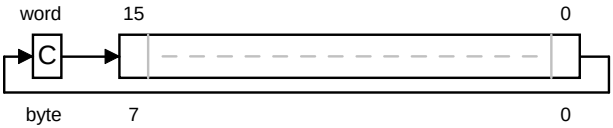
RRC dst or RRC.W dst
RRC dst

Operation

C -> MSB -> MSB-1 .... LSB+1 -> LSB -> C

Description

The destination operand is shifted right one position. The carry C is shifted into the MSB, the LSB is shifted into the carry C.



Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Set if initial destination is positive and initial Carry is set, otherwise reset

Mode Bits

OscOff, **CPUOff** and **GIE** are not affected

Example

R5 is shifted right one position. The MSB is loaded with 1.

```
SETC            ; PREPARE CARRY FOR MSB
RRC   R5        ; R5/2 + 8000h -> R5
```

Example

R5 is shifted right one position. The MSB is loaded with 1.

```
SETC            ; PREPARE CARRY FOR MSB
RRC.B R5        ; R5/2 + 80h -> R5; Low byte of R5 is used
```

B

*** SBC[.W]** Subtract borrow*) from destination
*** SBC.B** Subtract borrow*) from destination

Syntax
SBC dst or SBC.W dst
SBC.B dst

Operation
dst + 0FFFFh + C -> dst
dst + 0FFh + C -> dst

Emulation
SUBC #0,dst
SUBC.B #0,dst

Description The carry C is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Reset if dst was decremented from 0000 to 0FFFFh, set otherwise
Reset if dst was decremented from 00 to 0FFh, set otherwise
V: Set if initially C=0 and dst=08000h
Set if initially C=0 and dst=080h

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)     ; Subtract LSDs
SBC     2(R12)          ; Subtract carry from MSD
```

Example The 8bit counter pointed to by R13 is subtracted from a 16bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)     ; Subtract LSDs
SBC.B   1(R12)          ; Subtract carry from MSD
```

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

B

* SETC Set carry bit

Syntax SETC

Operation 1 -> C

Emulation BIS #1,SR

Description The Carry Bit C is set, an operation which is often necessary.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Emulation of the decimal subtraction:
Subtract R5 from R6 decimally
Assume that R5=3987 and R6=4137

```
DSUB    ADD     #6666h,R5       ; Move content R5 from 0-9 to 6-0Fh
                                ; R5 = 03987 + 6666 = 09FEDh
        INV     R5              ; Invert this(result back to 0-9)
                                ; R5 = .NOT. R5 = 06012h
        SETC                    ; Prepare carry = 1
        DADD    R5,R6           ; Emulate subtraction by adding of:
                                ; (10000 - R5 - 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 4137 + 06012 + 1 = 1 0150 = 0150
```

*** SETN** Set Negative bit

Syntax SETN

Operation 1 -> N

Emulation BIS #4,SR

Description The Negative bit N is set.

Status Bits
N: Set
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

*** SETZ** Set Zero bit

Syntax SETZ

Operation 1 -> Z

Emulation BIS #2,SR

Description The Zero bit Z is set.

Status Bits
N: Not affected
Z: Set
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

SUB[.W] subtract source from destination
SUB.B subtract source from destination

Syntax SUB src,dst or SUB.W src,dst
SUB.B src,dst

Operation dst + .NOT.src + 1 -> dst
or
[(dst - src -> dst)]

Description The source operand is subtracted from the destination operand. This is made by adding the 1's complement of the source operand and the constant 1. The source operand is not affected, the previous contents of the destination are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example See example at the SBC instruction

Example See example at the SBC.B instruction

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

B

SUBC[.W]SBB[.W] subtract source and borrow/.NOT. carry from destination
SUBC.B,SBB.B subtract source and borrow/.NOT. carry from destination

Syntax

SUBC	src,dst	or	SUBC.W	src,dst	or
SBB	src,dst	or	SBB.W	src,dst	
SUBC.B	src,dst	or	SBB.B	src,dst	

Operation

dst + .NOT.src + C -> dst
or
(dst - src - 1 + C -> dst)

Description The source operand is subtracted from the destination operand. This is made by adding of the 1's complement of the source operand and the carry C. The source operand is not affected, the previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset if not
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example Two floating point mantissas (24bits) are subtracted .
LSB's are in R13 resp. R10, MSB's are in R12 resp. R9.

```
SUB.W   R13,R10         ; 16bit part, LSB's
SUBC.B  R12,R9          ;  8bit part, MSB's
```

Example The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).

```
SUB.B   @R13+,R10   ; Subtract LSDs without carry
SUBC.B  @R13,R11    ; Subtract MSDs with carry
...                 ; resulting fron the LSDs
```

Note: Borrow is treated as a .NOT. carry

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

B

SWPB Swap bytes

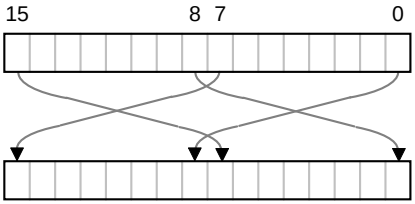
Syntax SWPB dst

Operation bits 15 to 8 <-> bits 7 to 0

Description The high and the low bytes of the destination operand are exchanged.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected



Example

```
MOV   #040BFh,R7        ; 0100000010111111 -> R7
SWPB  R7                ; 1011111101000000 in R7
```

Example The value in R5 is multiplied by 256. The result is stored in R5,R4

```
SWPB  R5                ;
MOV   R5,R4             ;Copy the swapped value to R4
BIC   #0FF00h,R5        ;Correct the result
BIC   #00FFh,R4         ;Correct the result
```

B

SXT Extend Sign

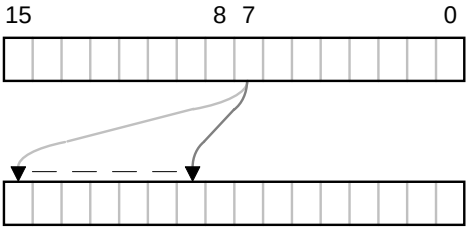
Syntax SXT dst

Operation Bit 7 -> Bit 8 ..... Bit 15

Description The sign of the Low byte is extended into the High byte.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected



Example R7 is loaded with Timer/Counter value. The operation of the sign extend instruction expands the bit8 to bit15 with the value of bit7.
R7 is added then to R6 where it is accumulated.

```
MOV.B   &TCDAT,R7    ; TCDAT = 080h: . . . .  . . . . 1000 0000
SXT     R7           ; R7 = 0FF80h:  1111 1111 1000 0000
ADD     R7,R6        ; add value of EDE to 16bit ACCU
```

B

*** TST[.W]** Test destination
*** TST.B** Test destination

Syntax

```
TST    dst    or    TST.W    dst
TST.B  dst
```

Operation

dst + 0FFFFh + 1
dst + 0FFh + 1

Emulation

```
CMP    #0,dst
CMP.B  #0,dst
```

Description The destination operand is compared to zero. The status bits are set according to the result. The destination is not affected.

Status Bits
N: Set if destination is negative, reset if positive
Z: Set if destination contains zero, reset otherwise
C: Set
V: Reset.

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example R7 is tested. If it is negative continue at R7NEG; if it is positive but not zero continue at R7POS.

```
        TST    R7        ; Test R7
        JN     R7NEG     ; R7 is negative
        JZ     R7ZERO    ; R7 is zero
R7POS   .....            ; R7 is positive but not zero

R7NEG   .....            ; R7 is negative

R7ZERO  .....            ; R7 is zero
```

Example Lowbyte of R7 is tested. If it is negative continue at R7NEG; if it is positive but not zero continue at R7POS.

```
        TST.B  R7        ; Test Low byte of R7
        JN     R7NEG     ; Low byte of R7 is negative
        JZ     R7ZERO    ; Low byte of R7 is zero
R7POS   .....            ; Low byte of R7 is positive but not zero

R7NEG   .....            ; Lowbyte of R7 is negative

R7ZERO  .....            ; Lowbyte of R7 is zero
```

B

XOR[.W] Exclusive OR of source with destination
XOR.B Exclusive OR of source with destination

Syntax XOR src,dst or XOR.W src,dst
XOR.B src,dst

Operation src .XOR. dst -> dst

Description The source operand and the destination operand are OR'ed exclusively. The result is placed into the destination. The source operand is not affected.

Status Bits
N: Set if MSB of result is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if both operands are negative

Mode Bits **OscOff**, **CPUOff** and **GIE** are not affected

Example The bits set in R6 toggle the bits in the RAM word TONI.

```
XOR     R6,TONI     ; Toggle bits of word TONI on the bits set in R6
```

Example The bits set in R6 toggle the bits in the RAM byte TONI.

```
XOR.B R6,TONI       ; Toggle bits in word TONI on bits
                    ; set in Low byte of R6,
```

Example Reset bits in Lowbyte of R7 to 0 that are different to bits in RAM byte EDE.

```
XOR.B   EDE,R7      ; Set different bit to '1s'
INV.B   R7          ; Invert Lowbyte, Highbyte is 0h
```

B

Macro instructions emulated with several instructions

The following table shows the instructions which need more words if emulated by the reduced instruction set. This is not of great concern, because they are rarely used. The immediate values -1, 0, +1, 2, 4 and 8 are provided by the Constant Generator Registers R2/CG1 and R3/CG2.

Emulated instruction			Instruction flow		Comment
ABS	**dst**		TST	dst	; Absolute value of destination
			JN	L$0	; Destination is negative
		L$1	...		; Destination is positive
			...		
			...		
		L$0	INV	dst	; Convert negative destination
			INC	dst	; to positive
			JMP	L$1	
DSUB	**src,dst**		ADD	#6666h,src	; Decimal subtraction
			INV	src	; Source is destroyed!
			SETC		
			DADD	src,dst	; DST - SRC (dec)
NEG	**dst**		INV	dst	; Negation of destination
			INC	dst	
RL	**dst**		ADD	dst,dst	; Rotate left circularly
			ADDC	#0,dst	
RR	**dst**		CLRC		; Rotate right circularly
			RRC	dst	
			JNC	L$1	
			BIS	#8000h,dst	
		L$1	...		