CALM
Common **A**ssembly **L**anguage for **M**icroprocessors

Instructions

Patrick Fäh, La Colombière, CH-1783 Pensier, Suisse

INSTRUCTIONS

An instruction is composed of an operation code which may be followed by operands. The operation code of an instruction is a mnemonic name, usually derived from an action verb expressing the operation to be performed.

Additional data specifiers completely determine the instruction.

The general syntax diagram of an instruction is given in fig. 25.

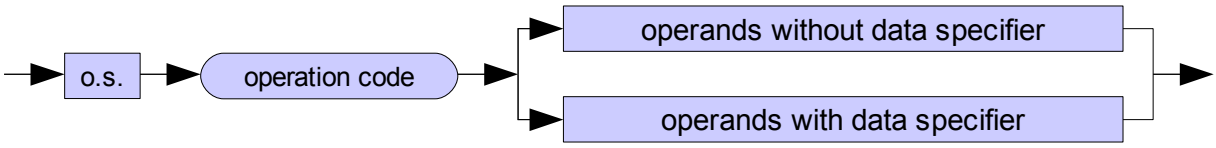
A few one-letter postfixes may be added to the CALM operation codes:

M	special instruction with multiple operands
P	special instruction involving some peripheral (I/O) action
I	indivisible instruction

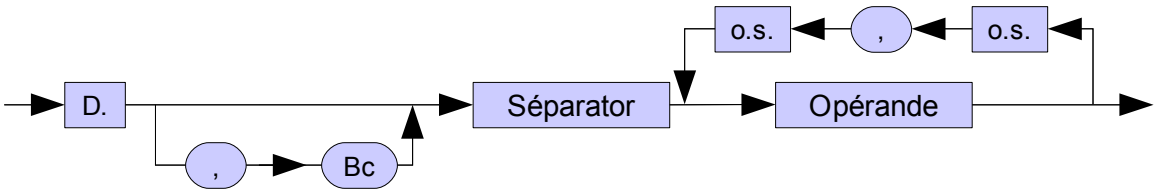
The condition code setting philosophy may vary from one processorto another and is not reflected in the following instructiondefinitions.

However, instructions which depart from the main processorphilosophy should not have the names defined in CALM.

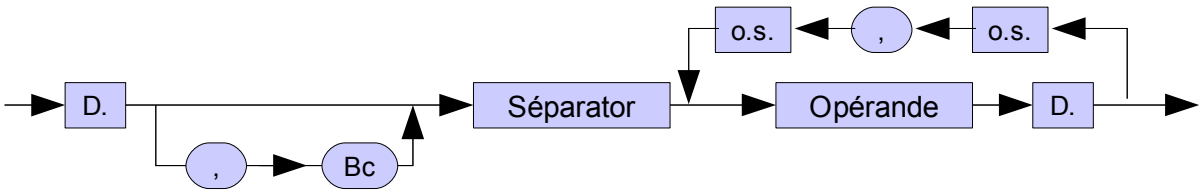
A short notation for data specifier is .ds in the following text.



Instruction



operands without data specifiers



operands with data specifiers

Bc.: condition code D.: data specifier

ACOC Add the 1's complement of the first operand, a one and

Description: Add the 1's complement of the first operand, a one and the carry bit C to the second operand. Operation is identical to SUBC but the flags are affected differently.

Syntax:
```
ACOC.ds   source, sourcedestination
ACOC.ds   source1, source2, destination
ACOC      source.ds, sourcedestination.ds
ACOC      source1.ds, source2.ds, destination.ds
```

Comment: This instruction exists only in older 8 bit processors (like SC/MP, 1802).

Example
```
:ACO      #16'4A,D          ; 1802
```

ADD ADD

Description.: Add two operands and update the flags.

Syntax:
```
ADD.ds    source, sourcedestination
ADD.ds    source1, source2, destination
ADD       source.ds, sourcedestination.ds
ADD       source1.ds, source2.ds, destination.ds
```

Examples
```
:ADD      {HL},A            ; 8080, Z80
ADD.16    {A6}-2,D0         ; 68000
ADD.F64   {R0}+10,{R1}+20   ; NS32000
```

ADDC ADD with Carry

Description: Add two operands and the carry bit C.

Syntax:
```
ADDC.ds   source, sourcedestination
ADDC.ds   source1, source2, destination
ADDC      source.ds, sourcedestination.ds
ADDC      source1.ds, source2.ds, destination.ds
```

Examples:
```
ADDC      {IX}+3,A          ; 6800, 680
ADDC.8    #1,[SS]+{BP}      ; iAPX86
ADDX.D8   {-A0},{-A1}       ; 68000, X is here C
```

AND AND

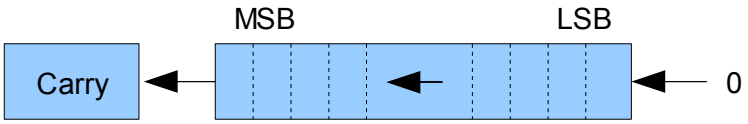
Description: Performs a bitwise logical AND of the two source.

Syntax:
```
AND.ds    source, sourcedestination
AND.ds    source1, source2, destination
AND       source.ds, sourcedestination.ds
AND       source1.ds, source2.ds, destination.ds
```

Comment: This instruction is used for masking bits. A similar instruction is BIC (bit clear), but rarely built-in.

Examples:
```
AND       #1,A              ; 8080, Z80,
AND.32    #16'FFFF0000,D6   ; 68000
BIC.16    #7,R0             ; NS32000 (AND.16 #16'FFF8,R0)
```

ASL Arithmetic Shift Left



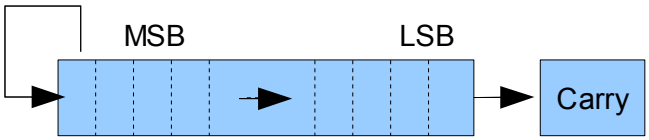
Description: This instruction shifts the source operand one or more places to the left (toward the MSB). The least significant bit(s) are replaced by zero(s). This instruction is the same as the SL instruction, but may act differently on the overflow bit V, which has to been set by this instruction if the sign bit of the result is not the same.

Syntax:
```
ASL.ds      sourcedestination
ASL.ds      amplitude, sourcedestination
ASL.ds      amplitude, source, destination
ASL         amplitude.ds, sourcedestination.ds
ASL         amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the right. ASL.16 sourcedestination is equivalent to SL.A16 sourcedestination. A shift to the left multiplies by two.

Example
```
: ASL.16    CL,BX                ; iAPX86
```

ASR Arithmetic Shift Right



Description: This instruction shifts the source operand one or more places to the right (toward the LSB), but the most significant bit is preserved and propagated to the right. The operation is identical to a divide by two for positive numbers (for negative numbers rounding is not made in the same direction).

Syntax:
```
ASR.ds      sourcedestination
ASR.ds      amplitude, sourcedestination
ASR.ds      amplitude, source, destination
ASR         amplitude.ds, sourcedestination.ds
ASR         amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the left. ASR.16 sourcedestination is equivalent to SR.A16 sourcedestination.

Examples:
```
ASR         B                    ; 6809 (ASR #1,B)
ASR.16      CL,BX                ; iAPX86
```

CALL CALL

Description: Save the return address on the stack and jump to the subroutine at the given address.

Syntax:
```
CALL    jumpaddress
CALL,c  jumpaddress
```

Comment: The called subroutine is terminated by the RET instruction.

Example:
```
CALL    ADRESS
```

CHECK CHECK

Description: Check a value against two bounds.

Syntax: CHECK.ds limite_inférieure, limite_supérieure, source

Comment: A double comparison is performed, but frequently one of the bound is the zero value and can be implicitly referred to. When the value is out of bound, important variations may occur: Some processors set only a flag and others generate a special check trap.

Example :
```
CHECK.A16  #100,D4          ; 68000 (0 <= D4 <= 100)
```

CLR CLeaR

Description: Clear the destination operand (set all bits to zero). Equivalent to the instruction: MOVE #0,destination.

Syntax: CLR.ds destination

Comment: If a CLR is conditional, the operand is not modified if the condition is false (see also SET).

Example :
```
CLR     {IX}+OFFSET:#BIT_NR ; Z80
CLR.32  D1                  ; 68000
CLRC
```

COMP COMPare

Description: Compare the second given operand to the first one. Same operation as SUB, but no result is generated (only the flags are updated).

Syntax:
```
COMP.ds  source1, source2
COMP     source1.ds, source2.ds
```

Examples:
```
COMP     {IX}+10,B          ; 6800,
COMP.F32 {SB}+4,F0          ; NS32000
```

CONV CONVert

Description: Perform a data type conversion. If source and destination are identical, the data specifiers may be appended to the operation code.

Syntax:
```
CONV.ds   source, destination
CONV      source.ds, destination.ds
```

Comment: One must specify the data type and the data size. Some data type conversion possibilities may be replaced by the MOVE instruction.

Examples :
```
CONV.A8.16 D0                ; 68000
CONV       {FP}.A16,{SB}.F64 ; NS32000
```

DEC DECrement

Description: Decrement by one the operand. Only one operand is llowed. A SUB instruction must be used for decrements by 2, 4, etc.

Syntax:
```
DEC.ds    sourcedestination
```

Examples :
```
DEC       {IX}+DEPLACEMENT   ; Z80
DEC.32    D1                 ; 68000
```

DIV DIVide

Description: Divide the second operand by the first one. The destination is either the second or the third operand. The remainder may be an additional last operand.

Syntax:
```
DIV.ds    diviseur, dividende, quotient, reste
DIV       diviseur.ds,dividende.ds,quotient.ds,reste.ds
```

Comment: An overflow may occur, depending on the size of the operands. This automatically generates a trap for some processors. The destination operand is very often identical to the source operand and sometimes also contains the remainder of the division in its upper part.

Examples:
```
DIV.16    CX,DXAX,AX,DX      ; iAPX86
DIV.A16   D1,D0              ; 68000 (reste: D0:#31..#16)
```

Note that: 5.DIV.2 = 2, -5.DIV.2 = -2, 5.DIV.-2 = -2, -5.DIV.-2 = 2. Et: A = (A.DIV.B)*B + (A.MOD.B), .MOD. = is the remainder of the division.DJ

DJ Decrement and Jump

Description: Decrement the first operand and jump to the address specified by the second operand.

Syntax: DJ.ds,c sourcedestination, addresse_de_saut

Comment: The most frequently implemented loop instruction decrements a counter and jumps to the given address, if a condition is met.

Examples:
```
DJ,NE     B,ADRESS           ; Z80
DJ.16,NMO D0,ADRESS          ; 68000 (NMO: pas moins un)
```

EX EXchange

Description: Exchange two operands. The order of the two operands is unimportant.

Syntax:
```
EX.ds     operand1, operand2
```

Comment: The instruction EX performs a simultaneous double move.

Examples:
```
EX      {SP}, HL          ; 8080, Z80
EX.16   AX, [DS] +SEMA    ; iAPX86
EX.32   A6, D0            ; 68000
```

HALT HALT

Description: Halt the processor until an external signal restarts it.

Syntax: HALT

Comment: After this instruction the processor releases the bus and even does not respond to interrupt requests. The processor can only be reactivated by an external reset signal. This instruction is rare. However, the HALT state is found in every modern microprocessor (i.e. double bus faults).

Example:
```
HALT                      ; PDP-11
```

INC INCrement

Description: Increment by one the operand. Only one operand is allowed. An ADD instruction must be used for increments by 2, 4, etc.

Syntax: INC.ds sourcedestination

Examples:
```
INC     {IX} +OFFSET      ; Z80
INC.32  D1                ; 68000
```

IOFF Interrupt OFF

Description: Interrupt off.

Syntax: IOFF

Comment: This instruction disables all hardware interrupts, except non maskable interrupts (NMI).

Examples:
```
IOFF                      ; 8080, Z80, 6800,
IOFF                      ; 68000 (OR.16 #16'0700,SF)
```

ION Interrupt ON,

Description: Interrupt on.

Syntax: ION

Examples:
```
ION                       ; 8080, Z80, 6800,
ION                       ; 68000 (AND.16 #16'F8FF,SF)
```

JUMP JUMP

Description: Jump to the given address.

Syntax:
```
JUMP      jumpaddress
JUMP,c    jumpaddress
```

Comment: The instruction JUMP ADDRESS is equivalent to MOVE #ADDRESS,PC.

Examples:
```
JUMP,VS   ADRESS                ; Z80, 6800
JUMP      R16^ADRESS            ; 6809, iAPX86,
JUMP      {{FP}+OFFSET1}+OFFSET2        ; NS32000
```

MOVE MOVE

Description: Move source operand to destination operand.

Syntax:
```
MOVE.ds   source, destination
MOVE      source.ds, destination.ds
```

Examples:
```
MOVE    HL,SP                   ; 8080, Z80
MOVE.8  #2,BL                   ; iAPX86
MOVE.32 #{A6}+OFFSET,A0         ; 68000
MOVE.F64 {SB}+OFFSET,F2         ; NS32000
```

MUL MULtiply

Description: Multiply the two operands. If the destination operand is not explicitly mentioned with its size, there will be an ambiguity in the size of the destination (double or single).

Syntax:
```
MUL.ds    source, sourcedestination
MUL.ds    source1, source2, destination
MUL       source.ds, sourcedestination.ds
MUL       source1.ds, source2.ds, destination.ds
```

Examples:
```
MUL       A,B,AB                ; 6801
MUL.16    D1,D0                 ; 68000 (MUL D1.16,D0.16,D0.32)
MUL.F64   {R0}+10,F2            ; NS32000
```

NEG NEGate

Description: Negate (2's complement or subtract from zero) the source operand.

Syntax:
```
NEG.ds    sourcedestination
NEG.ds    source, destination
NEG       source.ds, destination.ds
```

Examples:
```
NEG       A                     ; Z80,
NEG.16    {A6+}                 ; 68000
NEG.F64   {R0}+10,{R1}+20       ; NS32000
```

NEGC NEGate with Carry

Description: Negate with carry, this subtracts the carry bit and the operand from zero.

Syntax:
```
NEGC.ds
NEGC.ds    source, destination
NEGC       source.ds, destination.ds
```

Comment: The result of a NEGC operation is identical to a NOT operation, if the carry is initially set. However, the final value of the carry may be different.

Example:
```
NEGX.D8    D0                  ; 68000 (X is here C)
```

NOP No OPeration

Description: No operation.

Syntax: NOP

Comment: The microprocessor performs no operation; it is equivalent to a fetch with no action.

Example:
```
NOP
```

NOT NOT

Description: IPerform a bitwise complement (inversion) of the source operand (1's complement).

Syntax:
```
NOT.ds     sourcedestination
NOT.ds     source, destination
NOT        source.ds, destination.ds
```

Examples:
```
NOT        A                   ; 8080, Z80, ...
NOT.16     {A6+}               ; 68000
NOT.32     {R0}+10,{R1}+20     ; NS32000
```

OR OR

Description: Performs a bitwise logical OR of the two source operands.

Syntax:
```
OR.ds      source, sourcedestination
OR.ds      source1, source2, destination
OR         source.ds, sourcedestination.ds
OR         source1.ds, source2.ds, destination.ds
```

Comment: This instruction is used for setting bits. OR is sometimes called BIS (bit set), especially when the BIC (bit clear) instruction is available.

Examples:
```
OR         #1,A                ; 8080, Z80, ...
OR.32      #16'FFFF0000,D6     ; 68000
BIS.8      #3,F                ; NS32000 (OR.8 #16'3,F)
```

POP POP

Description: Pop from stack. For most present microprocessors: POP destination is equivalent to MOVE {SP+},destination.

Syntax: POP.ds destination

Comment: Usually, only one stack receives the subroutine return address. The instruction POP refers to that stack.

Examples:
```
POP      DE          ; 8080, Z80
POPU     AB          ; 6809 (de la pile US)
POP.16   F           ; 68010, 68020
```

PUSH PUSH

Description: Push on stack. For most present microprocessors: PUSH source is equivalent to MOVE source,{-SP}.

Syntax: PUSH.ds source

Comment: Usually, only one stack receives the subroutine return address. The instruction PUSH refers to that stack.

Examples:
```
PUSH     DE          ; 8080, Z80
PUSHU    AB          ; 6809 (de la pile US)
PUSH.16  F           ; 68010, 68020
```

RESET RESET

Description: Reset peripherals.

Syntax: RESET

Comment: The instruction RESET simulates the normal hardware reset signal by software (the RESET line is bidirectional). The processor itself is not reset.

Example:
```
RESET                68000
```

RET RETurn

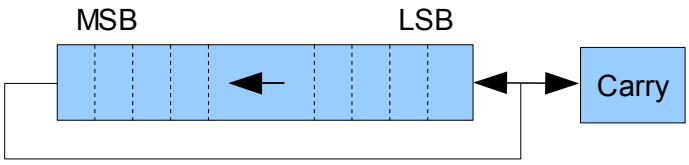
Description: Return from subroutine (address from stack).

Syntax:
```
RET
RET,c
RET          expression
```

Comment: The RET instruction terminates a subroutine (called from CALL). At the end of exception processing subroutines (like interrupts, traps), different RET instructions distinguish that other informations, apart from the program counter (PC), must be restored.

Examples:
```
RET,EQ               ; 8080, Z80
RETI                 ; Z80 (interruption)
RETSF                ; 68000 (PC + registre SF)
RET    #10           ; NS32000
```

RL Rotate Left



Description: This instruction causes the specified operand to be shifted one or more places to the left, with the LSB being replaced by the MSB on each shift.

Syntax:

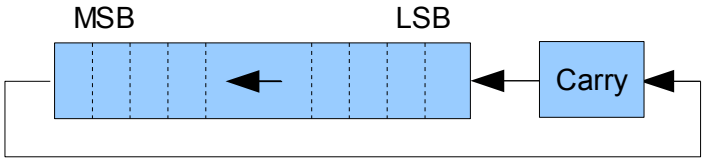
```
RL.ds       sourcedestination
RL.ds       amplitude, sourcedestination
RL.ds       amplitude, source, destination
RL          amplitude.ds, sourcedestination.ds
RL          amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the right.

Examples:

```
RL          B           ; 6809 (RL #1,B), ...
RL.16       CL,BX       ; iAPX86 (RL CL.8,BX.16)
```

RLC Rotate Left with Carry



Description: This instruction causes the specified operand to be shifted one or more places to the left, with the LSB being replaced by the carry, and the carry replaced by the MSB on each shift.

Syntax:

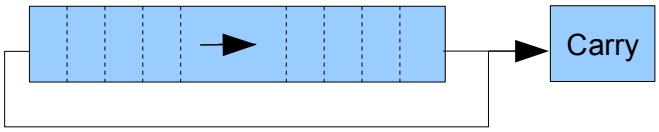
```
RLC.ds      sourcedestination
RLC.ds      amplitude, sourcedestination
RLC.ds      amplitude, source, destination
RLC         amplitude.ds, sourcedestination.ds
RLC         amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the right.

Examples:

```
RLC         B           ; 6809 (RLC #1,B), ...
RLC.16      CL,BX       ; iAPX86 (RLC CL.8,BX.16)
```

RR Rotate Right



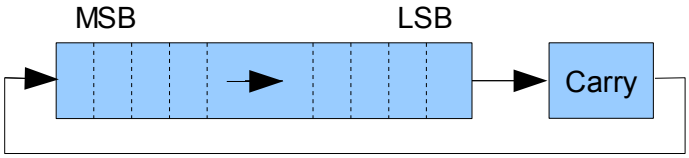
Description: This instruction causes the specified operand to be shifted one or more places to the right, with the MSB being replaced by the LSB on each shift.

Syntax:
```
RR.ds       sourcedestination
RR.ds       amplitude, sourcedestination
RR.ds       amplitude, source, destination
RR          amplitude.ds, sourcedestination.ds
RR          amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the left.

Examples:
```
RR          B               ; 6809 (RR #1,B), ...
RR.16       CL,BX           ; iAPX86 (RR CL.8,BX.16)
```

RRC Rotate Right with Carry



Description: This instruction causes the specified operand to be shifted one or more places to the right, with the MSB being replaced by the carry, and the carry by the LSB on each shift.

Syntax:
```
RRC.ds      sourcedestination
RRC.ds  amplitude, sourcedestination
RRC.ds  amplitude, source, destination
RRC     amplitude.ds, sourcedestination.ds
RRC     amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the left.

Examples:
```
RRC         B               ; 6809 (RRC #1,B), ...
RRC.16      CL,BX           ; iAPX86 (RRC CL.8,BX.16)
```

SET SET

Description: Set the destination operand (set all bits to one). Equivalent to the instruction MOVE #-1,destination.

Syntax: SET.ds destination

Comment: If a SET instruction is conditional, the operand is set (all ones) if the condition is true, and cleared (all zeroes) if the condition is false (refer also to CLR).

Examples:
```
SET.8,EQ    {A0}            ; 68000
SET         {HL}:#4         ; Z80
SETC                        ; 8080, Z80, ...
```

SKIP SKIP

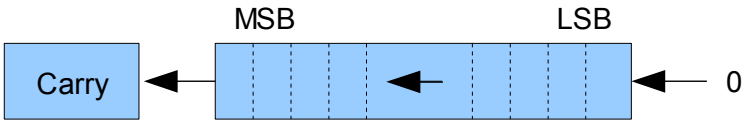
Description: Skip next instruction.

Syntax: SKIP,c

Comment: The SKIP instruction is very often combined with a condition.

Example: SKIP,EQ DJ.16,NMO D0,ADRESS ; 68000

SL Shift Left



Description: This instruction causes the specified operand to be shifted one or more places to the left, with the LSB being replaced by zero on each shift.

Syntax:

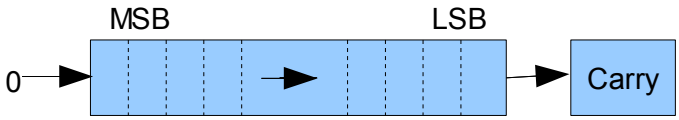
```
SL.ds     sourcedestination
SL.ds     amplitude, sourcedestination
SL.ds     amplitude, source, destination
SL        amplitude.ds, sourcedestination.ds
SL        amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the right. A shift to the left multiplies by two.

Examples:

```
SL        B           ; 6809 (SL #1,B), ...
SL.16     CL,BX       ; iAPX86 (SL CL.8,BX.16)
```

SR Shift Right



Description: This instruction causes the specified operand to be shifted one or more places to the right, with the MSB being replaced by zero on each shift.

Syntax:

```
SR.ds     sourcedestination
SR.ds     amplitude, sourcedestination
SR.ds     amplitude, source, destination
SR        amplitude.ds, sourcedestination.ds
SR        amplitude.ds, source.ds, destination.ds
```

Comment: A negative shift means, that the shift direction is to the left. Shift right divides by two.

Examples:

```
SR        B           ; 6809 (SR #1,B), ...
SR.16     CL,BX       ; iAPX86 (SR CL.8,BX.16)
```

SUB SUBtract

Description: Subtract the first operand from the second operand.

Syntax:
```
SUB.ds    source, sourcedestination
SUB.ds    source1, source2, destination
SUB       source.ds, sourcedestination.ds
SUB       source1.ds, source2.ds, destination.ds
```

Examples:
```
SUB       #2'1010,A          ; 8080, Z80, ...
SUB.32    {A6}+400,D6        ; 68000
SUB.F64   F0,{SB}+10         ; NS32000
```

SUBC SUBtract with Carry

Description: Subtract the first operand and the carry bit C from the second operand.

Syntax:
```
SUBC.ds source, sourcedestination
SUBC.ds source1, source2, destination
SUBC    source.ds, sourcedestination.ds
SUBC    source1.ds, source2.ds, destination.ds
```

Examples:
```
SUBC     #2'1010,A          ; 8080, Z80,
SUBX.32  {-A4},{-A3}        ; 68000 (X est ici C)
SUBC.32  R0,{SB}+10         ; NS32000
```

SWAP SWAP

Description: The two halves of the operand are swapped. The associated size is the size of the full operand.

Syntax: SWAP.ds sourcedestination

Comment: The SWAP instruction is necessary, when the subunits of a operand are not accessible independently.

Example:
```
SWAP.32 D4                   ; 68000
```

TCLR Test and CLeaR

Description: Test and clear the operand bit(s) to zero.

Syntax:
```
TCLR.ds   sourcedestination  (:ADRESS_de_bit(.ds))
TCLR      sourcedestination.ds (:ADRESS_de_bit(.ds))
```

Comment: The zero bit flag Z often contains the bit value before the instruction is executed. The additional letter I specifies, that the operation is indivisible. The data specifier appended to the operation code or the byte address operand indicates the bit addressing limits.

Examples:
```
TCLR.32   D5:#31             ; 68000 (bit: 0..31)
TCLR.8    {A6}+CONDITION:#2  ; 68000 (bit: 0..7)
TCLRI     {SB}:{FP}.A8       ; NS32000 (bit: -128..+127)
```

TEST TEST

Description: Test the sign and zero value of the operand.

Syntax:
```
TEST.ds    source (:ADRESS_de_bit(.ds))
TEST       source.da (:ADRESS_de_bit(.ds))
TEST.ds    source1, source2
```

Comment: TEST operand is equivalent to COMP #0,operand. The zero bit flag is set, if the operand value is zero. The MSB is copied in the sign bit flag N. For two operands an AND operation is performed before the test.

Examples:
```
TEST       B                    ; 6800,
TEST.32    D5:#31               ; 68000 (bit: 0..31)
TEST.8     {A6}+CONDITION:#2    ; 68000 (bit: 0..7)
TEST       {SB}:{FP}.A8         ; NS32000 (bit: -128..+127)
TEST.16    [DS]+{SI},AX         ; iAPX86
```

TNOT Test and NOT

Description: Test and complement the operand bit(s).

Syntax:
```
TNOT.ds    sourcedestination  (:ADRESS_de_bit(.ds))
TNOT       sourcedestination.ds (:ADRESS_de_bit(.ds))
```

Comment: The zero bit flag Z often contains the bit value before the execution of the instruction. The data specifier appended to the operation code or the byte address operand indicates the bit addressing limits.

Examples:
```
TNOT.32    D5:#31               ; 68000 (bit: 0..31)
TNOT.8     {A6}+CONDITION:#2    ; 68000 (bit: 0..7)
TNOT       {SB}:{FP}.A8         ; NS32000 (bit: -128..+127)
```

TRAP TRAP

Description: Special subroutine call.

Syntax:
```
TRAP       expression
TRAP,c     expression
```

Comment: TRAPs may be caused by software or by hardware (like divide by zero).

Examples:
```
TRAP                            ; 6800,
TRAP       #15                  ; 68000
TRAP,VS                         ; 68000
```

TSET Test and SET

Description: Test and set the operand bit(s) to zero.

Syntax:
```
TSET.ds     sourcedestination    (:ADRESS_de_bit(.ds))
TSET        sourcedestination.ds (:ADRESS_de_bit(.ds))
```

Comment: The zero bit flag Z often contains the bit value before the instruction is executed. The additional letter I specifies, that the operation is indivisible. The data specifier appended to the operation code or the byte address operand indicates the bit addressing limits.

Examples:
```
TSET.32     D5:#31              ; 68000 (bit: 0..31)
TSET.8      {A6}+CONDITION:#2   ; 68000 (bit: 0..7)
TSETI       {SB}:{FP}.A8        ; NS32000 (bit: -128..+127)
```

WAIT WAIT

Description: Wait for interrupt.

Syntax: WAIT

Comment: The WAIT instruction minimizes the response time for an interrupt request. After an interrupt, and after the execution of the interrupt subroutine, the processor executes the next instruction after the WAIT instruction. There are four variant types of WAIT: WAIT (CPU waits actively), SLEEP (the current for the I/O is cut off), STOP (the internal oszillator is stopped) and HALT (CPU stops).

Example: WAIT

XOR XOR

Description: Performs a bitwise logical XOR of the two source operands.

Syntax:
```
XOR.ds      source, sourcedestination
XOR.ds      source1, source2, destination
XOR         source.ds, sourcedestination.ds
XOR         source1.ds, source2.ds, destination.ds
```

Comment: This instruction is used for complementing bits.

Examples:
```
XOR         #1,A                ; 8080, Z80
XOR.32      #16'FFFF0000,D6     ; 68000
```

Fin du document.