Megaprocessor

--

Instruction Set

James Newman
May 2016

Introduction

This document describes the instruction set for the Megaprocessor. They are listed in alphabetical order (by assemble mnemonic).

The following abbreviations are used:

cc	Condition Code. Coded by a 4 bit code in the range 2..15.
RA	One of the general purpose registers used as a destination. Coded with a 2 bit value.
RB	One of the general purpose registers used as a source. Coded with a 2 bit value.
RC	Either R0 or R1 may be source or destination. Coded with a 1 bit value.
RI	Either R2 or R3 used as index. Coded with a 1 bit value: 0 for R2, 1 for R3.
displ	8 bit signed displacement. Ranges -128..127.
dir	Direction. Coded with a 1 bit value. 0 = load (memory to Reg), 1 = store (Reg to memory)
sz	Operand size. Coded with a 1 bit value. 1=Byte, 0=Word
sgn	Selects signed/unsigned operation. Coded with a 1 bit value. 0 = unsigned, 1 = signed.

Note the assembler syntax is

```
        instruction         destination, source
```

Summary

Group:	MOVER	AND	XOR	OR	ADD	ADDQ	SUB	CMP
	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x00	sxt r0	test r0	xor r0,r0	inv r0	add r0,r0	addq r0,#2	neg r0	abs r0
0x01	move r1,r0	and r1,r0	xor r1,r0	or r1,r0	add r1,r0	addq r1,#2	sub r1,r0	cmp r1,r0
0x02	move r2,r0	and r2,r0	xor r2,r0	or r2,r0	add r2,r0	addq r2,#2	sub r2,r0	cmp r2,r0
0x03	move r3,r0	and r3,r0	xor r3,r0	or r3,r0	add r3,r0	addq r3,#2	sub r3,r0	cmp r3,r0
0x04	move r0,r1	and r0,r1	xor r0,r1	or r0,r1	add r0,r1	addq r0,#1	sub r0,r1	cmp r0,r1
0x05	sxt r1	test r1	xor r1,r1	inv r1	add r1,r1	addq r1,#1	neg r1	abs r1
0x06	move r2,r1	and r2,r1	xor r2,r1	or r2,r1	add r2,r1	addq r2,#1	sub r2,r1	cmp r2,r1
0x07	move r3,r1	and r3,r1	xor r3,r1	or r3,r1	add r3,r1	addq r3,#1	sub r3,r1	cmp r3,r1
0x08	move r0,r2	and r0,r2	xor r0,r2	or r0,r2	add r0,r2	addq r0,#-2	sub r0,r2	cmp r0,r2
0x09	move r1,r2	and r1,r2	xor r1,r2	or r1,r2	add r1,r2	addq r1,#-2	sub r1,r2	cmp r1,r2
0x0A	sxt r2	test r2	xor r2,r2	inv r2	add r2,r2	addq r2,#-2	neg r2	abs r2
0x0B	move r3,r2	and r3,r2	xor r3,r2	or r3,r2	add r3,r2	addq r3,#-2	sub r3,r2	cmp r3,r2
0x0C	move r0,r3	and r0,r3	xor r0,r3	or r0,r3	add r0,r3	addq r0,#-1	sub r0,r3	cmp r0,r3
0x0D	move r1,r3	and r1,r3	xor r1,r3	or r1,r3	add r1,r3	addq r1,#-1	sub r1,r3	cmp r1,r3
0x0E	move r2,r3	and r2,r3	xor r2,r3	or r2,r3	add r2,r3	addq r2,#-1	sub r2,r3	cmp r2,r3
0x0F	sxt r3	test r3	xor r3,r3	inv r3	add r3,r3	addq r3,#-1	neg r3	abs r3

Group:	Indirect	PostInc	Stack rel	Absolute	Push/Pop	Immediate	Branch	Misc
	0x80	0x90	0xA0	0xB0	0xC0	0xD0	0xE0	0xF0
0x00	ld.w r0,(r2)	ld.w r0,(r2++)	ld.w r0,(sp+m)	ld.w r0,addr	pop r0	ld.w r0,#data	buc dd	move r0,sp
0x01	ld.w r1,(r2)	ld.w r1,(r2++)	ld.w r1,(sp+m)	ld.w r1,addr	pop r1	ld.w r1,#data	bus dd	move sp,r0
0x02	ld.w r0,(r3)	ld.w r0,(r3++)	ld.w r2,(sp+m)	ld.w r2,addr	pop r2	ld.w r2,#data	bhi dd	jmp (r0)
0x03	ld.w r1,(r3)	ld.w r1,(r3++)	ld.w r3,(sp+m)	ld.w r3,addr	pop r3	ld.w r3#data	bls dd	jmp addr
0x04	ld.b r0,(r2)	ld.b r0,(r2++)	ld.b r0,(sp+m)	ld.b r0,addr	pop ps	ld.b r0,#data	bcc dd	andi ps,#data
0x05	ld.b r1,(r2)	ld.b r1,(r2++)	ld.b r1,(sp+m)	ld.b r1,addr	(nop)	ld.b r1,#data	bcs dd	ori ps,#data
0x06	ld.b r0,(r3)	ld.b r0,(r3++)	ld.b r2,(sp+m)	ld.b r2,addr	ret	ld.b r2,#data	bne dd	add.b sp,#data
0x07	ld.b r1,(r3)	ld.b r1,(r3++)	ld.b r3,(sp+m)	ld.b r3,addr	reti	ld.b r3#data	beq dd	sqrt
0x08	st.w (r2),r0	st.w (r2++),r0	st.w (sp+m),r0	st.w addr,r0	push r0	shift r0,descr	bvc dd	mulu
0x09	st.w (r2),r1	st.w (r2++),r1	st.w (sp+m),r1	st.w addr,r1	push r1	shift r1,descr	bvs dd	muls
0x0A	st.w (r3),r0	st.w (r3++),r0	st.w (sp+m),r2	st.w addr,r2	push r2	shift r2,descr	bpl dd	divu
0x0B	st.w (r3),r1	st.w (r3++),r1	st.w (sp+m),r3	st.w addr,r3	push r3	shift r3,descr	bmi dd	divs
0x0C	st.b (r2),r0	st.b (r2++),r0	st.b (sp+m),r0	st.b addr,r0	push ps	bit r0,descr	bge dd	addx r0,r1
0x0D	st.b (r2),r1	st.b (r2++),r1	st.b (sp+m),r1	st.b addr,r1	trap #n	bit r1,descr	blt dd	subx r0,r1
0x0E	st.b (r3),r0	st.b (r3++),r0	st.b (sp+m),r2	st.b addr,r2	jsr (r0)	bit r2,descr	bgt dd	negx r0
0x0F	st.b (r3),r1	st.b (r3++),r1	st.b (sp+m),r3	st.b addr,r3	jsr addr	bit r3,descr	ble dd	nop

ABS RA

Operation:

```
if (RA < 0)
        - RA    ⇒ RA
end if
```

Description:
The contents of the general purpose register are replaced by the absolute. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0x7, CMP				RA		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	?	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	?	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	0?	Set if carry is generated. Cleared otherwise

Example:

```
R0[8000] R1[D4CC] R2[2B35] R3[A6AB] PC[007D] SP[233B] PS[71(CX...ID.)]
007C: 65     :ABS   R0

R0[8000] R1[D4CC] R2[2B35] R3[A6AB] PC[007E] SP[233B] PS[7B(CXV.NID.)]
007D: 65     :ABS   R1

R0[8000] R1[2B34] R2[2B35] R3[A6AB] PC[007F] SP[233B] PS[71(CX...ID.)]
007E: 65     :ABS   R2

R0[8000] R1[2B34] R2[2B35] R3[A6AB] PC[0080] SP[233B] PS[41(.....ID.)]
```

ADD RA, RB

Operation:

RA + RB ⇒ RA

Description:
The contents of the source general purpose register are added to the destination general purpose register. The condition codes are set according to the result. The source and destination registers may be the same.

Format:

7	6	5	4	3	2	1	0
Code = 0x4, ADD				RB		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

Example:
```
R0[0000] R1[848B] R2[FFFF] R3[A6AB] PC[0075] SP[233B] PS[43(....NID.)]
0074: 4D     :ADD   R1,R3

R0[0000] R1[2B36] R2[FFFF] R3[A6AB] PC[0076] SP[233B] PS[79(CXV..ID.)]
```

ADD SP, #immediate data

Operation:

SP + data ⇒ SP

Description:
The immediate value is added to the stack pointer. The immediate value is an 8 bit signed value.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	1	1	0
Imm(7:0)							

Length/Cycles:
2 bytes; 2 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:

```
R0[010C] R1[002B] R2[552B] R3[5678] PC[011F] SP[8321] PS[23(C...NI..)]
011E: F6 12     :ADD       SP, #0x12

R0[010C] R1[002B] R2[552B] R3[5678] PC[0121] SP[8333] PS[23(C...NI..)]
0120: F6 EE     :ADD       SP, #0xEE

R0[010C] R1[002B] R2[552B] R3[5678] PC[0123] SP[8321] PS[23(C...NI..)]
```

ADDQ RA, #small

Operation:

RA + immediate $\Rightarrow$ RA

Description:
The immediate operand is added to the destination general purpose. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0x5, ADDQ				small code		RA	

Small code

00	2
01	1
10	-2
11	-1

INC RA is an alias for ADDQ R1, #1
DEC RA is an alias for ADDQ R1, #-1

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

Example:
```
R0[0000] R1[2B36] R2[FFFF] R3[A6AB] PC[0076] SP[233B] PS[79(CXV..ID.)]
0075: 4D     :ADDQ  R1,#-2

R0[0000] R1[2B34] R2[FFFF] R3[A6AB] PC[0077] SP[233B] PS[71(CX...ID.)]
```

ADDX R0, R1

Operation:

R0 + R1 + X $\Rightarrow$ R0

Description:
The contents of R0, R1 and the X flag are added and the result is stored in R0. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				1	1	0	0

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Cleared if result is non-zero, otherwise unchanged
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

NB. If the Z flag is set before the start of an operation then it will achieve correct test for a zero result after completing an extended precision operation.

Example:
```
R0[151A] R1[002B] R2[552B] R3[5678] PC[00F8] SP[8321] PS[10(.X......)]
00F7: FC     :ADDX  R0,R1

R0[1546] R1[002B] R2[552B] R3[5678] PC[00F9] SP[8321] PS[00(........)]
00F8: FC     :ADDX  R0,R1

R0[1571] R1[002B] R2[552B] R3[5678] PC[00FA] SP[8321] PS[00(........)]
```

AND RA, RB

Operation:

RA & RB ⇒ RA

Description:
The bitwise AND of the source and destination general purpose registers is stored in the destination. The condition codes are set according to the result. The source and destination registers must be different.

Format:

7	6	5	4	3	2	1	0
Code = 0x1, AND				RB		RA	

RA ≠ RB

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[F059] R1[00A2] R2[4826] R3[7E04] PC[67CA] SP[233B] PS[C0(......DU)]
67C9: 1B     :AND    R3, R2

R0[F059] R1[00A2] R2[4826] R3[4804] PC[67CB] SP[233B] PS[C2(....N.DU)]
```

AND PS, #immediate data

Operation:

PS & data ⇒ PS

Description:
The immediate value is ANDed with the PS register.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	1	0	0
Imm(7:0)							

Length/Cycles:
2 bytes; 2 cycles

Condition Codes:

I	*	Set as per result of operation
N	*	Set as per result of operation
Z	*	Set as per result of operation
V	*	Set as per result of operation
X	*	Set as per result of operation
C	*	Set as per result of operation

Example:

```
R0[015E] R1[002B] R2[552B] R3[5678] PC[00EC] SP[0068] PS[27(C..ZNI..)]
00EB: F4 DE     :AND      PS, #0xDE

R0[015E] R1[002B] R2[552B] R3[5678] PC[00EE] SP[0068] PS[06(...ZN...)]
```

Arithmetic Shift: ASL, ASR

Operation:

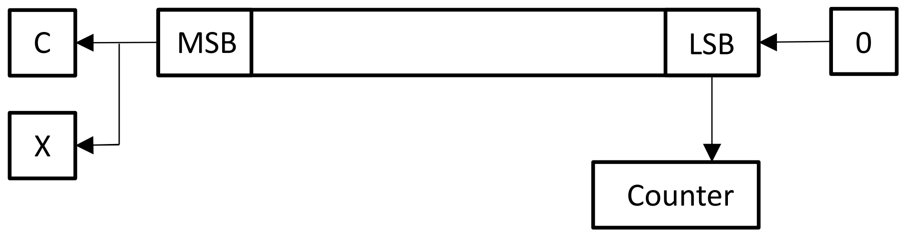
RA shifted by amount	⇒	RA
weight of n bits of RA	⇒	RA

Description:
The specified general register is shifted in the given direction for the required number of bits. The last bit shifted out is copied to C and X. For left shifts the input bit is 0. For right shifts the input bit is a replication of the sign bit of the register.
The shift amount may be specified in two ways:

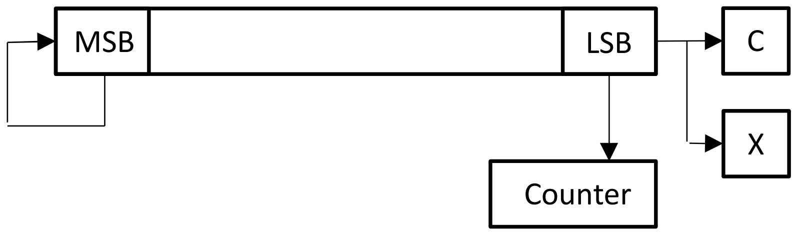
1. as an immediate operand, this is a 5 bit signed value. (In the assembler use of the ASR instruction is translated to ASL and the immediate value is negated to compensate).
2. as a register, Rp, in which case the least significant 5 bits are used and treated as a signed value. These 5 bits are negated if the ASR instruction was used.

If the shift count is specified by a register then the instruction may be qualified with .WT. In this case the number of 1 bits that are shifted out of the least significant bit of the operand are counted and the result stored in RA.

ASL:



ASR:



Format:

ASd RA, #n_to_shift

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 01, arithmetic		I/R = 0	n_to_shift				

ASd RA, Rp
ASd.WT RA, Rp

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 01, arithmetic		I/R = 1	L/R	W	0	Rp	

Encoding:

	0	1
I/R	immediate	register
L/R	left	right
W	shift	weight

Length/Cycles:
2 bytes, 4 + 1 per shift cycles

Condition Codes – weight not selected:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if MSB changes at any time during shift, cleared otherwise
X	*	Set according to last bit shifted out of operand, not affected for a shift of zero.
C	*	Set according to last bit shifted out of operand, cleared for a shift of zero.

Condition Codes – weight selected:

I	-	Not affected
N	0	Always cleared
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	0	Always cleared
C	*	Set according to least significant bit of result, i.e. parity of the n bits of the operand.

Example:
```
R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01A4] SP[8321] PS[02(....N...)]
01A3: D8 40       :ASL     R0,#0

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01A6] SP[8321] PS[00(........)]
01A5: D8 42       :ASL     R0,#2

R0[48D0] R1[9ABC] R2[9AFD] R3[FFFE] PC[01A8] SP[8321] PS[00(........)]
01A7: D8 63       :ASL     R0,R3

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01AA] SP[8321] PS[00(........)]
01A9: D8 73       :ASR     R0,R3

R0[48D0] R1[9ABC] R2[9AFD] R3[FFFE] PC[01AC] SP[8321] PS[00(........)]
01AB: D9 62       :ASL     R1, R2

R0[48D0] R1[F357] R2[9AFD] R3[FFFE] PC[01AE] SP[8321] PS[32(CX..N...)]
01AD: D9 6A       :ASL.WT  R1, R2

R0[48D0] R1[0003] R2[9AFD] R3[FFFE] PC[01B0] SP[8321] PS[20(C.......)]
01AF: D9 6A       :ASL.WT  R1, R2

R0[48D0] R1[0002] R2[9AFD] R3[FFFE] PC[01B2] SP[8321] PS[00(........)]
```

Bcc target

Operation:

```
if (condition true)
        PC + displacement        ⇒ PC
end if
```

Description:
The status flags are tested according to the condition. If the condition is met then the displacement is added to the program counter. The displacement is an 8 bit signed value.

Format:

7	6	5	4	3	2	1	0
Code = 0xE, branch				cc			
displacement							

Mnemonic	Condition	Coding	Test
UC	User Clear	0x00	~U
US	User Set	0x01	U
HI	High	0x02	(~C) & (~Z)
LS	Low or Same	0x03	C | Z
CC/HS	Carry Clear	0x04	~C
CS/LO	Carry Set	0x05	C
NE	Not Equal	0x06	~Z
EQ	Equal	0x07	Z
VC	Overflow Clear	0x08	~V
VS	Overflow Set	0x09	V
PL	Plus	0x0A	~N
MI	Minus	0x0B	N
GE	Greater or Equal	0x0C	(N & V) | (~N & ~V)
LT	Less Than	0x0D	(N & ~V) | (~N & V)
GT	Greater Than	0x0E	(N & V & ~Z) | (~N & ~V & ~Z)
LE	Less Or Equal	0x0F	Z | (N & ~V) | (~N & V)

Length/Cycles:
2 bytes, 3 cycles if branch taken, 2 cycles if not

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:

```
R0[015E] R1[002B] R2[552B] R3[5678] PC[00E3] SP[0068] PS[23(C...NI..)]
00E2: E4 01    :BCC 0x00E5

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E5] SP[0068] PS[23(C...NI..)]
00E4: E5 02    :BCS 0x00E8

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E9] SP[0068] PS[23(C...NI..)]
00E8: FF       :NOP

R0[015E] R1[002B] R2[552B] R3[5678] PC[00EA] SP[0068] PS[23(C...NI..)]
00E9: E5 FD    :BCS 0x00E8

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E9] SP[0068] PS[23(C...NI..)]
```

BCHG

Operation:

~(<bit number> of RA) $\Rightarrow$ <bit number> of RA

Description:
A bit in the specified general register is tested and its value used to update the Z flag (if it is 0 then Z is set, if it is 1 then Z is cleared). The bit is then inverted. The bit number may be specified as an immediate value, or from the bottom four bits of a general purpose register (which may be the same one).

Format:

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 01		I/R = 0	0	bit_number			

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 01		I/R = 1	0	0	0	Rp	

Encoding:

	0	1
I/R	immediate	register

Length/Cycles:
2 bytes, 3 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	*	Set if specified bit is zero, cleared otherwise
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[0002] R1[0008] R2[0006] R3[5678] PC[0139] SP[8321] PS[10(.X......)]
0138: DE 60       :BCHG    R2, R0

R0[0002] R1[0008] R2[0002] R3[5678] PC[013B] SP[8321] PS[10(.X......)]
013A: DE 61       : BCHG    R2, R1

R0[0002] R1[0008] R2[0102] R3[5678] PC[013D] SP[8321] PS[14(.X.Z....)]
013C: DE 41       : BCHG    R2, #1

R0[0002] R1[0008] R2[0100] R3[5678] PC[013F] SP[8321] PS[10(.X......)]
```

BCLR

Operation:

0 ⇒ <bit number> of RA

Description:
A bit in the specified general register is tested and its value used to update the Z flag (if it is 0 then Z is set, if it is 1 then Z is cleared). The bit is then set to 0. The bit number may be specified as an immediate value, or from the bottom four bits of a general purpose register (which may be the same one).

Format:

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 10		I/R = 0	0	bit_number			

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 10		I/R = 1	0	0	0	Rp	

Encoding:

	0	1
I/R	immediate	register

Length/Cycles:
2 bytes, 3 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	*	Set if specified bit is zero, cleared otherwise
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[0002] R1[0008] R2[0006] R3[5678] PC[0129] SP[8321] PS[10(.X......)]
0128: DE A0       :BCLR    R2, R0

R0[0002] R1[0008] R2[0002] R3[5678] PC[012B] SP[8321] PS[10(.X......)]
012A: DE A1       :BCLR    R2, R1

R0[0002] R1[0008] R2[0002] R3[5678] PC[012D] SP[8321] PS[14(.X.Z....)]
012C: DE 81       :BCLR    R2, #1

R0[0002] R1[0008] R2[0000] R3[5678] PC[012F] SP[8321] PS[10(.X......)]
```

BSET

Operation:

1 ⇒ <bit number> of RA

Description:
A bit in the specified general register is tested and its value used to update the Z flag (if it is 0 then Z is set, if it is 1 then Z is cleared). The bit is then set to 1. The bit number may be specified as an immediate value, or from the bottom four bits of a general purpose register (which may be the same one).

Format:

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 11		I/R = 0	0	bit_number			

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 11		I/R = 1	0	0	0	Rp	

Encoding:

	0	1
I/R	immediate	register

Length/Cycles:
2 bytes, 3 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	*	Set if specified bit is zero, cleared otherwise
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[0002] R1[0008] R2[0006] R3[5678] PC[0131] SP[8321] PS[10(.X......)]
0130: DE E0       :BSET    R2, R0

R0[0002] R1[0008] R2[0106] R3[5678] PC[0133] SP[8321] PS[10(.X......)]
0132: DE E1       : BSET    R2, R1

R0[0002] R1[0008] R2[0106] R3[5678] PC[0135] SP[8321] PS[14(.X.Z....)]
0134: DE C1       : BSET    R2, #1

R0[0002] R1[0008] R2[0106] R3[5678] PC[0137] SP[8321] PS[10(.X......)]
```

BTST

Operation:

test <bit number> of RA

Description:
A bit in the specified general register is tested and its value used to update the Z flag (if it is 0 then Z is set, if it is 1 then Z is cleared). The bit number may be specified as an immediate value, or from the bottom four bits of a general purpose register (which may be the same one).

Format:

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 00		I/R = 0	0	bit_number			

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	1	RA	
fn = 00		I/R = 1	0	0	0	Rp	

Encoding:

	0	1
I/R	immediate	register

Length/Cycles:
2 bytes, 3 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	*	Set if specified bit is zero, cleared otherwise
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[0002] R1[0008] R2[0006] R3[5678] PC[0141] SP[8321] PS[10(.X......)]
0140: DE 20       :BTST    R2, R0

R0[0002] R1[0008] R2[0006] R3[5678] PC[0143] SP[8321] PS[10(.X......)]
0142: DE 21       : BTST    R2, R1

R0[0002] R1[0008] R2[0006] R3[5678] PC[0145] SP[8321] PS[14(.X.Z....)]
0144: DE 01       : BTST    R2, #1

R0[0002] R1[0008] R2[0006] R3[5678] PC[0147] SP[8321] PS[10(.X......)]
```

CMP RA, RB

Operation:

RA - RB

Description:
The contents of the source general purpose register are subtracted from the destination general purpose register. The condition codes are set according to the result, and the result discarded. The source and destination registers must be different.

Format:

7	6	5	4	3	2	1	0
Code = 0x7, CMP				RB		RA	

RA ≠ RB

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if overflow is generated. Cleared otherwise
X	*	Not affected
C	*	Set if carry is generated. Cleared otherwise

Example:
```
R0[8000] R1[2B34] R2[2B35] R3[A6AB] PC[0080] SP[233B] PS[41(.....ID.)]
007F: 65     :CMP   R1,R3

R0[8000] R1[2B34] R2[2B35] R3[A6AB] PC[0081] SP[233B] PS[6B(C.V.NID.)]
```

DIVS
DIVU

Operation:

```
if (R1 == 0)
        execute TRAP 1
else
        Quotient(R0 / R1)        ⇒ R2
        Remainder(R0 / R1)       ⇒ R3
end if
if (signed operation)
        abs( R1 )        ⇒ R1
end if
```

Description:
If R1 is zero then TRAP 2 is taken. Otherwise R0 / R1 is calculated. The quotient is put in R2, the remainder. The arithmetic condition flags are cleared.
There are two ways of defining signed division, the difference lying in whether or not negative remainders are allowed. Both are implemented, the method used is controlled by the D flag.

	D = 0	D = 1
13/3	Q = 4, R = 1	Q = 4, R = 1
13/-3	Q = -4, R = 1	Q = -4, R = 1
-13/3	Q = -4, R = -1	Q = -5, R = 2
-13/-3	Q = 4, R = -1	Q = 5, R = 2

In all cases the following holds:
Dividend = (Quotient × Divisor) + Remainder

Format:
DIVU
DIVS

7	6	5	4	3	2	1	0
0xF, miscellaneous				1	0	1	sgn

Encoding:

	0	1
sgn	unsigned	signed

Length/Cycles:
1 byte, division by zero is 7 cycles, unsigned is 18 cycles, signed is 19 cycles

Condition Codes:

I	-	Not affected
N	0	Set to zero
Z	0	Set to zero
V	0	Set to zero
X	0	Set to zero
C	0	Set to zero

Example:

```
R0[000D] R1[FFFD] R2[DAC9] R3[0237] PC[0162] SP[8321] PS[02(....N...)]
0161: FA            :DIV.U

R0[000D] R1[FFFD] R2[0000] R3[000D] PC[0163] SP[8321] PS[00(........)]
0162: FA            :DIV.S

R0[000D] R1[0003] R2[FFFC] R3[0001] PC[0164] SP[8321] PS[00(........)]
0163: D0 F3 FF      :LD.W  R0,0xFFF3

R0[FFF3] R1[0003] R2[FFFC] R3[0001] PC[0167] SP[8321] PS[02(....N...)]
0166: FA            :DIV.S

R0[FFF3] R1[0003] R2[FFFC] R3[FFFF] PC[0168] SP[8321] PS[00(........)]
0167: FA 40         :OR    PS,#0x40

R0[FFF3] R1[0003] R2[FFFC] R3[FFFF] PC[016A] SP[8321] PS[40(......D.)]
0169: FA            :DIV.S

R0[FFF3] R1[0003] R2[FFFB] R3[0002] PC[016B] SP[8321] PS[40(......D.)]
```

INV RA

Operation:

~RA ⇒ RA

Description:
The general purpose register is set to its bitwise inverse. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0x3, OR				RA		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[0000] R1[0001] R2[FFFF] R3[2223] PC[0073] SP[233B] PS[41(.....ID.)]
0072: 35     :INV   R1

R0[0000] R1[848B] R2[FFFF] R3[7B75] PC[0074] SP[233B] PS[43(....NID.)]
```

JMP (R0)

Operation:

R0 ⇒ PC

Description:
The program counter is set to the value in R0.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	0	1	0

Length/Cycles:
1 byte; 2 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[010C] R1[002B] R2[552B] R3[5678] PC[0109] SP[0068] PS[01(.....I..)]
0108: F2     :JMP (R0)

R0[010C] R1[002B] R2[552B] R3[5678] PC[010D] SP[0068] PS[01(.....I..)]
```

JMP absolute

Operation:

address $\Rightarrow$ PC

Description:
The program counter is set to the value of the address operand.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	0	1	1
Absolute(7:0)							
Absolute(15:8)							

Length/Cycles:
3 bytes; 4 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:

```
R0[015E] R1[002B] R2[552B] R3[5678] PC[0110] SP[0068] PS[01(.....I..)]
010F: F3 18 01   :JMP 0x0118

R0[015E] R1[002B] R2[552B] R3[5678] PC[0119] SP[0068] PS[01(.....I..)]
```

JSR (R0)

Operation:

SP – 2	⇒ SP
PC + 1	⇒ [SP]
R0	⇒ PC

Description:
The stack pointer is decremented by two. The address of the next instruction is stored in memory at the location pointed to by the stack pointer. The program counter is set to the value in R0.

Format:

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				1	1	1	0

Length/Cycles:
1 byte; 4 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
0060: 00 78 00 78 56 FF 34 2B 55 FF 55 DE 55 55 DE FF

R0[015E] R1[002B] R2[552B] R3[5678] PC[00DD] SP[0068] PS[01(.....I..)]
00DC: CE     :JSR (R0)

R0[015E] R1[002B] R2[552B] R3[5678] PC[015F] SP[0066] PS[01(.....I..)]

0060: 00 78 00 78 56 2B DD 00 55 FF 55 DE 55 55 DE FF
```

JSR absolute

Operation:

SP – 2 ⇒ SP
PC + 1 ⇒ [SP]
address ⇒ PC

Description:
The stack pointer is decremented by two. The address of the next instruction is stored in memory at the location pointed to by the stack pointer. The program counter is set to the value of the address operand.

Format:

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				1	1	1	1
Absolute(7:0)							
Absolute(15:8)							

Length/Cycles:
3 bytes; 6 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
0060: 00 78 00 78 56 FF 34 2B 55 FF 55 DE 55 55 DE FF

R0[015E] R1[002B] R2[552B] R3[5678] PC[00DD] SP[0068] PS[01(.....I..)]
00DC: CF 5E 01   :JSR 0x015E

R0[015E] R1[002B] R2[552B] R3[5678] PC[015F] SP[0066] PS[01(.....I..)]

0060: 00 78 00 78 56 2B DF 00 55 FF 55 DE 55 55 DE FF
```

LD immediate data

Operation:

source ⇒ destination

Description:
The immediate data is copied into the destination general purpose register. Data transfers may be byte or word. Words in memory are formatted little-endian. Bytes loaded from memory are zero extended to fill the destination register.

Format:

LD.SZ RA, #immediate

SZ may be B or W

7	6	5	4	3	2	1	0
Code = 0xD, immediate				0	sz=0	RA	
immediate(7:0)							
immediate (15:8)							

7	6	5	4	3	2	1	0
Code = 0xD, immediate				0	sz=1	RA	
immediate (7:0)							

Encoding:

	0	1
sz	word	byte

Length/Cycles:
2 bytes, 2 cycles for a byte transfer; 3 bytes, 3 cycles for a word transfer

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[0031] R1[DE55] R2[00DE] R3[5678] PC[00AC] SP[0068] PS[41(.....ID.)]
00AB: D7 6A     :LD.B      R3, 0x006A

R0[0031] R1[DE55] R2[00DE] R3[006A] PC[00AE] SP[0068] PS[41(.....ID.)]
00AD: D0 CD AB  :LD.W      R0, 0xABCD

R0[ABCD] R1[DE55] R2[00DE] R3[006A] PC[00B1] SP[0068] PS[43(....NID.)]
```

LD, ST indirect addressing

Operation:

source ⇒ destination

Description:
The index register (which may be R2 or R3) points to a memory location. For a load instruction the contents of the memory location memory are copied to the destination general purpose register which may be R0 or R1. For a store instruction the contents of the source register which may be R0 or R1 are copied to memory. Data transfers may be byte or word. Words in memory are formatted little-endian. Bytes loaded from memory are zero extended to fill the destination register.

Format:

LD.SZ RC, (RI)
ST.SZ (RI), RC

SZ may be B or W
RC may be R0 or R1
RI may be R2 or R3

7	6	5	4	3	2	1	0
Code = 0x8, indirect				dir	sz	ri	rc

Encoding:

	0	1
dir	LD	ST
sz	word	byte
ri	R2	R3
rc	R0	R1

Length/Cycles:
1 byte; 2 cycles for a byte transfer, three cycles for a word transfer

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
0060: 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF

R0[8000] R1[2B34] R2[0064] R3[0066] PC[00A6] SP[233B] PS[41(.....ID.)]
00A5: 8D     :ST.B  (R2), R1

R0[8000] R1[2B34] R2[0064] R3[0066] PC[00A7] SP[233B] PS[41(.....ID.)]
00A6: 8B     :ST.W  (R3), R1

R0[8000] R1[2B34] R2[0064] R3[0066] PC[00A8] SP[233B] PS[41(.....ID.)]
00A7: 86     :LD.B  R0,(R3)

R0[0034] R1[2B34] R2[0064] R3[0066] PC[00A9] SP[233B] PS[41(.....ID.)]
00A8: 82     :LD.W  R0,(R3)

R0[2B34] R1[2B34] R2[0064] R3[0066] PC[00AA] SP[233B] PS[41(.....ID.)]

0060: 00 00 00 00 34 FF 34 2B FF FF FF FF FF FF FF FF
```

LD, ST post-increment addressing

Operation:

source ⇒ destination
index register ⇒ index register + size_of_transfer

Description:
The index register (which may be R2 or R3) points to a memory location. For a load instruction the contents of the memory location memory are copied to the destination general purpose register which may be R0 or R1. For a store instruction the contents of the source register which may be R0 or R1 are copied to memory. Data transfers may be byte or word. After the transfer has completed the index register is incremented by one for each byte transferred. Words in memory are formatted little-endian. Bytes loaded from memory are zero extended to fill the destination register.

Format:

LD.SZ RC, (RI)
ST.SZ (RI), RC

SZ may be B or W
RC may be R0 or R1
RI may be R2 or R3

7	6	5	4	3	2	1	0
Code = 0x9, post increment				dir	sz	ri	rc

Encoding:

	0	1
dir	LD	ST
sz	word	byte
ri	R2	R3
rc	R0	R1

Length/Cycles:
1 byte; 2 cycles for a byte transfer, three cycles for a word transfer

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
0060: 00 00 00 00 34 FF 34 2B FF FF FF FF FF FF FF FF

R0[ABCD] R1[DE55] R2[0068] R3[006A] PC[00B4] SP[233B] PS[43(....NID.)]
00B3: 9D      :ST.B  (R2++), R1

R0[ABCD] R1[DE55] R2[0069] R3[006A] PC[00B5] SP[233B] PS[43(....NID.)]
00B4: 9B      :ST.W  (R3++), R1

R0[ABCD] R1[DE55] R2[0064] R3[006C] PC[00B6] SP[233B] PS[43(....NID.)]
00B5: 94      :LD.B  R0,(R2++)

R0[00FF] R1[DE55] R2[006A] R3[006C] PC[00B7] SP[233B] PS[41(.....ID.)]
00B6: 90      :LD.W  R0,(R2++)

R0[DE55] R1[DE55] R2[006C] R3[006C] PC[00B8] SP[233B] PS[43(....NID.)]

0060: 00 00 00 00 34 FF 34 2B 55 FF 55 DE FF FF FF FF
```

LD, ST stack relative addressing

Operation:

source ⇒ destination

Description:
An address is formed as the sum of the stack pointer and an 8 bit unsigned modifier. For a load instruction the contents of the memory location memory are copied to the destination general purpose register. For a store instruction the contents of the source general purpose register are copied to memory. Data transfers may be byte or word. Words in memory are formatted little-endian. Bytes loaded from memory are zero extended to fill the destination register.

Format:

LD.SZ RA, (SP, mm)
ST.SZ (SP, mm), RA

SZ may be B or W

7	6	5	4	3	2	1	0
Code = 0xA, stack relative				dir	sz	RA	
mm(7:0)							

Encoding:

	0	1
dir	LD	ST
sz	word	byte

Length/Cycles:
2 bytes; 3 cycles for a byte transfer, 4 cycles for a word transfer

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
0060: 00 00 00 00 34 FF 34 2B 55 FF 55 DE FF FF FF FF

R0[0031] R1[DE55] R2[006C] R3[006C] PC[00BD] SP[0068] PS[41(.....ID.)]
00BC: AD 04  :ST.B  (SP, 4), R1

R0[0031] R1[DE55] R2[006C] R3[006C] PC[00BF] SP[0068] PS[43(....NID.)]
00BE: A9 05  :ST.W  (SP, 5), R1

R0[0031] R1[DE55] R2[006C] R3[006C] PC[00C1] SP[0068] PS[43(....NID.)]
00C0: A6 03  :LD.B  R2,(SP, 3)

R0[0031] R1[DE55] R2[00DE] R3[006C] PC[00C3] SP[0068] PS[41(.....ID.)]
00C2: A3 03   :LD.W R3,(SP, 3)

R0[0031] R1[DE55] R2[00DE] R3[55DE] PC[00C5] SP[0068] PS[41(.....ID.)]

0060: 00 00 00 00 34 FF 34 2B 55 FF 55 DE 55 55 DE FF
```

LD, ST absolute addressing

Operation:

source ⇒ destination

Description:
The address points to a memory location. For a load instruction the contents of the memory location memory are copied to the destination general purpose register. For a store instruction the contents of the source general purpose register are copied to memory. Data transfers may be byte or word. Words in memory are formatted little-endian. Bytes loaded from memory are zero extended to fill the destination register.

Format:

LD.SZ RA, address
ST.SZ address, RA

SZ may be B or W

7	6	5	4	3	2	1	0
Code = 0xB, absolute				dir	sz	RA	
address(7:0)							
address(15:8)							

Encoding:

	0	1
dir	LD	ST
sz	word	byte

Length/Cycles:
3 bytes; 4 cycles for a byte transfer, 5 cycles for a word transfer

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
0060: 00 00 00 00 34 FF 34 2B 55 FF 55 DE 55 55 DE FF

R0[0031] R1[DE55] R2[00DE] R3[5678] PC[00CA] SP[0068] PS[41(.....ID.)]
00C9: BF 61 00  :ST.B      0x0061, R3

R0[0031] R1[DE55] R2[00DE] R3[5678] PC[00CD] SP[0068] PS[41(.....ID.)]
00CC: BB 63 00  :ST.W      0x0063, R3

R0[0031] R1[DE55] R2[00DE] R3[5678] PC[00D0] SP[0068] PS[41(.....ID.)]
00CF: B5 67 00  :LD.B      R1,0x0067

R0[0031] R1[002B] R2[00DE] R3[5678] PC[00D3] SP[0068] PS[41(.....ID.)]
00D2: B2 67 00  :LD.W      R2,0x0067

R0[0031] R1[002B] R2[552B] R3[5678] PC[00D6] SP[0068] PS[41(.....ID.)]

0060: 00 78 00 78 56 FF 34 2B 55 FF 55 DE 55 55 DE FF
```

Logical Shift: LSL, LSR

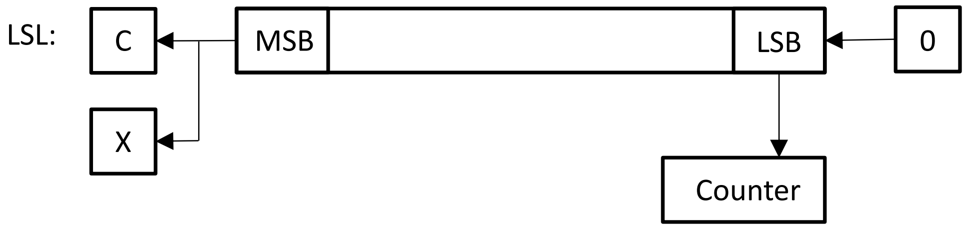
Operation:

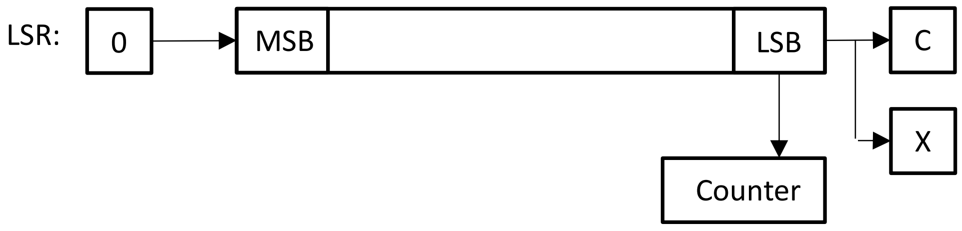
RA shifted by amount ⇒ RA
weight of n bits of RA ⇒ RA

Description:
The specified general register is shifted in the given direction for the required number of bits. The last bit shifted out is copied to C and X. For both left and right shifts the input bit is 0.
The shift amount may be specified in two ways:

1. as an immediate operand, this is a 5 bit signed value. (In the assembler use of the LSR instruction is translated to LSL and the immediate value is negated to compensate).
2. as a register, Rp, in which case the least significant 5 bits are used and treated as a signed value. These 5 bits are negated if the LSR instruction was used.

If the shift count is specified by a register then the instruction may be qualified with .WT. In this case the number of 1 bits that are shifted out of the least significant bit of the operand are counted and the result stored in RA.





Format:

LSd RA, #n_to_shift

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 00, logical		I/R = 0	n_to_shift				

LSd RA, Rp
LSd.WT RA, Rp

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 00, logical		I/R = 1	L/R	W	0	Rp	

Encoding:

	0	1
I/R	immediate	register
L/R	left	right
W	shift	weight

Length/Cycles:
2 bytes, 4 + 1 per shift cycles

Condition Codes – weight not selected:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if MSB changes at any time during shift, cleared otherwise
X	*	Set according to last bit shifted out of operand, not affected for a shift of zero.
C	*	Set according to last bit shifted out of operand, cleared for a shift of zero.

Condition Codes – weight selected:

I	-	Not affected
N	0	Always cleared
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	0	Always cleared
C	*	Set according to least significant bit of result, i.e. parity of the n bits of the operand.

Example:
```
R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[018A] SP[8321] PS[02(....N...)]
0189: D8 00       :LSL     R0,#0

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[018C] SP[8321] PS[00(........)]
018B: D8 02       :LSL     R0,#2

R0[48D0] R1[9ABC] R2[9AFD] R3[FFFE] PC[018E] SP[8321] PS[00(........)]
018D: D8 23       :LSL     R0,R3

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[0190] SP[8321] PS[00(........)]
018F: D8 33       :LSR     R0,R3

R0[48D0] R1[9ABC] R2[9AFD] R3[FFFE] PC[0192] SP[8321] PS[00(........)]
0191: D9 22       :LSL     R1, R2

R0[48D0] R1[1357] R2[9AFD] R3[FFFE] PC[0194] SP[8321] PS[30(CX......)]
0193: D9 2A       :LSL.WT  R1, R2

R0[48D0] R1[0003] R2[9AFD] R3[FFFE] PC[0196] SP[8321] PS[20(C.......)]
0195: D9 2A       :LSL.WT  R1, R2

R0[48D0] R1[0002] R2[9AFD] R3[FFFE] PC[0198] SP[8321] PS[00(........)]
```

MOVE RA, RB

Operation:

RB ⇒ RA

Description:
The contents of the source general purpose register are copied to the destination general purpose register. The condition codes are set according to the data transferred. Only 16 bits transfers are possible. The source and destination registers must be different.

Format:

7	6	5	4	3	2	1	0
Code = 0x0, MOVE				RB		RA	

RA ≠ RB

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[F059] R1[18ED] R2[41FB] R3[A3C0] PC[67CA] SP[233B] PS[F2(CX..N.DU)]
67C9: 01     :MOVE    R1, R0

R0[F059] R1[F059] R2[41FB] R3[A3C0] PC[67CB] SP[233B] PS[D2(.X..N.DU)]
```

Data Flow:

n	St.	PC	Instr	Addr.	Data	ALU 1	ADD 2	H	RA	RB			
	end	N		N	[N]		PC + 1 = N + 1						
0	0/end	N+1	[N]	N+1	[N+1]		PC +1 = N + 2		XX	YY			
	0	N+2	[N+1]	N+2	[N+2]				XX	XX			

MOVE SP/R0

Operation:

source ⇒ destination

Description:
Either SP is moved to R0, or the reverse depending on the instruction.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	0	0	mv

Encoding:

	0	1
mv	R0,SP	SP,R0

Length/Cycles:
1 bytes; 2 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:

```
R0[015E] R1[002B] R2[552B] R3[5678] PC[00EE] SP[0068] PS[06(...ZN...)]
00ED: F0         :MOVE     R0, SP

R0[0068] R1[002B] R2[552B] R3[5678] PC[00EF] SP[0068] PS[06(...ZN...)]
00EE: D0 21 83   :MOVE.W   R0, #0x8321

R0[8321] R1[002B] R2[552B] R3[5678] PC[00F2] SP[0068] PS[02(....N...)]
00F1: F1         :AND      SP, R0

R0[8321] R1[002B] R2[552B] R3[5678] PC[00F3] SP[8321] PS[06(....N...)]
```

MULS, MULU

Operation:

```
LSW(R0×R1)    ⇒ R2
MSW(R0×R1)    ⇒ R3
if (signed multiplication)
        if (R0 <0)
                0       ⇒ R0
        end if
        if (R1 <0)
                R0 + R1 ⇒ R0
        end if
end if
```

Description:
The product of R0 and R1 is calculated. The least significant 16 bits of the result are placed in R2, the most significant 16 bits are placed in R3. The arithmetic condition flags are cleared. If the operation is signed multiplication then R0 will be replaced by the sum of those contents of R0 and R1 that are negative.

Format:

MULU
MULS

7	6	5	4	3	2	1	0
0xF, miscellaneous				1	0	0	sgn

Encoding:

	0	1
sgn	unsigned	signed

Length/Cycles:
1 byte, unsigned is 18 cycles, signed is 19 cycles

Condition Codes:

I	-	Not affected
N	0	Set to zero
Z	0	Set to zero
V	0	Set to zero
X	0	Set to zero
C	0	Set to zero

Example:

```
R0[0123] R1[CAFE] R2[0006] R3[0000] PC[0151] SP[8321] PS[00(........)]
0150: F8       :MULU

R0[0123] R1[CAFE] R2[BEBA] R3[00E6] PC[0152] SP[8321] PS[00(........)]
0151: F9       :MULS

R0[0123] R1[CAFE] R2[BEBA] R3[FFC3] PC[0153] SP[8321] PS[00(........)]
```

NEG RA

Operation:

-RA ⇒ RA

Description:
The contents of the source general purpose register are negated. The condition codes are set according to the result. Applying this instruction to a register with the value of 0x8000 will produce a result of 0x8000 and set the overflow flag.

Format:

7	6	5	4	3	2	1	0
Code = 0x6, SUB				RA		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

Example:

```
R0[8000] R1[2B34] R2[FFFF] R3[A6AB] PC[007A] SP[233B] PS[53(.X..NID.)]
0079: 65     :NEG   R1

R0[8000] R1[D4CC] R2[FFFF] R3[A6AB] PC[007B] SP[233B] PS[73(CX..NID.)]
007A: 65     :NEG   R2

R0[8000] R1[D4CC] R2[0001] R3[A6AB] PC[007C] SP[233B] PS[71(CX...ID.)]
007B: 65     :NEG   R0

R0[8000] R1[D4CC] R2[0001] R3[A6AB] PC[007D] SP[233B] PS[7B(CXV.NID.)]
```

NEGX R0

Operation:

0 - R0 - X ⇒ R0

Description:
The contents R0 and the X flag are subtracted from 0 and the result is stored in R0. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				1	1	1	0

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Cleared if result is non-zero, otherwise unchanged
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

NB. If the Z flag is set before the start of an operation then it will achieve correct test for a zero result after completing an extended precision operation.

Example:
```
R0[151A] R1[002B] R2[552B] R3[5678] PC[0104] SP[8321] PS[00(........)]
0103: FE     :NEGX  R0,R1

R0[EAE6] R1[002B] R2[552B] R3[5678] PC[0105] SP[8321] PS[32(CX..N...)]
0104: FE     :NEGX  R0,R1

R0[1519] R1[002B] R2[552B] R3[5678] PC[0106] SP[8321] PS[30(CX......)]
```

NOP

Operation:

none

Description:
This instruction does nothing.
Currently opcode 0xC5 is not used and so is also mapped to NOP.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				1	1	1	1

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				0	1	0	1

Length/Cycles:
1 bytes; 1 cycle

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
R0[010C] R1[002B] R2[552B] R3[5678] PC[011E] SP[8321] PS[23(C...NI..)]
011D: FF       :NOP

R0[010C] R1[002B] R2[552B] R3[5678] PC[011F] SP[8321] PS[23(C...NI..)]
```

OR RA, RB

Operation:

RA | RB ⇒ RA

Description:
The bitwise or of the source and destination general purpose registers is written to the destination register. The condition codes are set according to the result. The source and destination registers must be different.

Format:

7	6	5	4	3	2	1	0
Code = 0x3, OR				RB		RA	

RA ≠ RB

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[0000] R1[848B] R2[FFFF] R3[7B75] PC[0074] SP[233B] PS[43(....NID.)]
0073: 37     :OR    R3,R1

R0[0000] R1[848B] R2[FFFF] R3[A6AB] PC[0075] SP[233B] PS[43(....NID.)]
```

OR PS, #immediate data

Operation:

PS || data ⇒ PS

Description:
The immediate value is ORed with the PS register.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				0	1	0	1
Imm(7:0)							

Length/Cycles:
2 bytes; 2 cycles

Condition Codes:

I	*	Set as per result of operation
N	*	Set as per result of operation
Z	*	Set as per result of operation
V	*	Set as per result of operation
X	*	Set as per result of operation
C	*	Set as per result of operation

Example:

```
R0[015E] R1[002B] R2[552B] R3[5678] PC[00EA] SP[0068] PS[23(C...NI..)]
00E9: F5 04     :OR       PS, #0x04

R0[015E] R1[002B] R2[552B] R3[5678] PC[00EC] SP[0068] PS[27(C..ZNI..)]
```

PUSH, POP

Operation:

source ⇒ destination

Description:
For a push instruction the stack pointer is decremented by one for a byte sized register (PS) or by two for a word register (RA). The register is then stored at the address pointed to by the stack pointer.
For a pop instruction the register is loaded from the address pointed to by the stack pointer. The stack pointer is then incremented by one for the byte sized register (PS) or by two for a word sized register (RA).

Format:

PUSH RA
POP RA

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				dir	0	RA	

PUSH PS
POP PS

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				dir	1	0	0

Encoding:

	0	1
dir	POP	PUSH

Length/Cycles:
1 byte; 2 cycles for a byte transfer (PS), 3 cycles for a word transfer (RA)

Condition Codes:
For PUSH/POP RA :

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

For PUSH PS:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

For POP PS:

I	*	Loaded from memory
N	*	Loaded from memory
Z	*	Loaded from memory
V	*	Loaded from memory
X	*	Loaded from memory
C	*	Loaded from memory

Example:

```
0060: 00 78 00 78 56 FF 34 2B 55 FF 55 DE 55 55 DE FF

R0[0031] R1[002B] R2[552B] R3[5678] PC[00D6] SP[0068] PS[41(.....ID.)]
00D5: CC     :PUSH  PS

R0[0031] R1[DE55] R2[006C] R3[006C] PC[00D7] SP[0067] PS[41(.....ID.)]
00D6: CA     :PUSH R2

R0[0031] R1[DE55] R2[006C] R3[006C] PC[00D8] SP[0065] PS[41(.....ID.)]
00D7: C4     :POP PS

R0[0031] R1[DE55] R2[00DE] R3[006C] PC[00D9] SP[0066] PS[2B(C.V.NI..)]
00D8: C0     :POP R0

R0[4155] R1[DE55] R2[00DE] R3[55DE] PC[00DA] SP[0068] PS[01(.....I..)]

0060: 00 78 00 78 56 2B 55 41 55 FF 55 DE 55 55 DE FF
```

RET

Operation:

[SP] $\Rightarrow$ PC
SP + 2 $\Rightarrow$ SP

Description:
The word in memory pointed to by the stack pointer is loaded into the program counter. The stack pointer is then incremented by 2. This is the complement of the JSR instructions and is used to return from a subroutine.

Format:

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				0	1	1	0

Length/Cycles:
1 byte; 4 cycles

Condition Codes:

I	-	Not affected
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
0060: 00 78 00 78 56 2B DF 00 55 FF 55 DE 55 55 DE FF

R0[015E] R1[002B] R2[552B] R3[5678] PC[015F] SP[0066] PS[01(.....I..)]
015E: C6     :RET

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E0] SP[0068] PS[01(.....I..)]

0060: 00 78 00 78 56 2B DF 00 55 FF 55 DE 55 55 DE FF
```

RETI

Operation:

[SP]	$\Rightarrow$	PS
SP + 1	$\Rightarrow$	SP
[SP]	$\Rightarrow$	PC
SP + 2	$\Rightarrow$	SP

Description:
The byte pointed to by the stack pointer is loaded into PS. The stack pointer is then incremented by one. The word in memory pointed to by the stack pointer is loaded into the program counter. The stack pointer is then incremented by 2. This is the complement of the TRAP and interrupt operations and is used to return from an exception handler.

Format:

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				0	1	1	1

Length/Cycles:
1 byte; 5 cycles

Condition Codes:

I	*	Loaded from memory
N	*	Loaded from memory
Z	*	Loaded from memory
V	*	Loaded from memory
X	*	Loaded from memory
C	*	Loaded from memory

Example:
```
0060: 00 78 00 78 56 01 E0 00 55 FF 55 DE 55 55 DE FF

R0[015E] R1[002B] R2[552B] R3[5678] PC[015F] SP[0065] PS[00(........)]
000C: C7     :RETI

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E1] SP[0068] PS[01(.....I..)]

0060: 00 78 00 78 56 01 E0 00 55 FF 55 DE 55 55 DE FF
```

Rotate: ROL, ROR

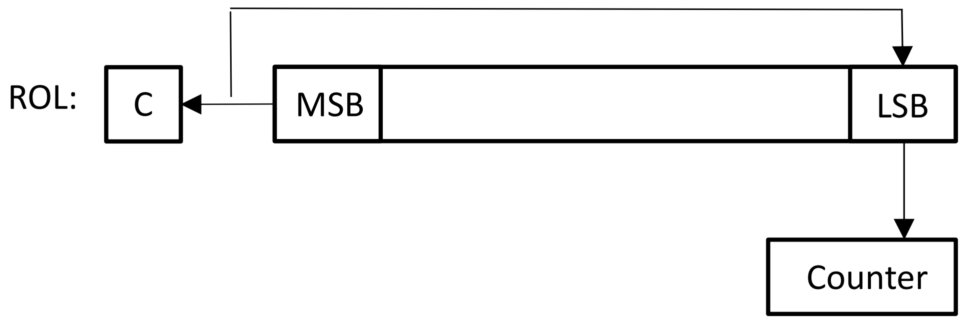
Operation:

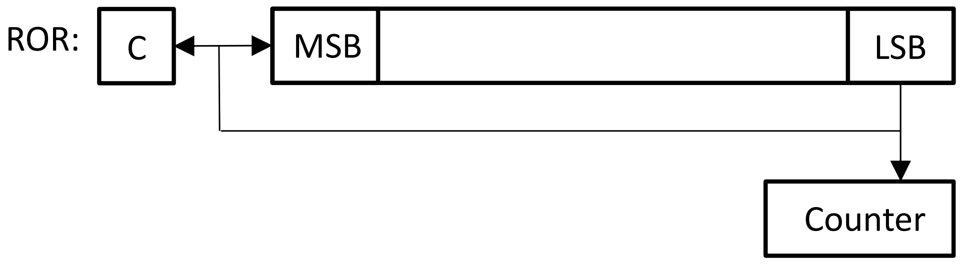
RA rotated by amount ⇒ RA
weight of n bits of RA ⇒ RA

Description:
The specified general register is rotated in the given direction for the required number of bits. The last bit rotated out of the operand (MSB if rotating left, LSB if rotating right) copied to C. X is not affected. The shift amount may be specified in two ways:

1. as an immediate operand, this is a 5 bit signed value. (In the assembler use of the ROR instruction is translated to ROL and the immediate value is negated to compensate).
2. as a register, Rp, in which case the least significant 5 bits are used and treated as a signed value. These 5 bits are negated if the ROR instruction was used.

If the shift count is specified by a register then the instruction may be qualified with .WT. In this case the number of 1 bits that are rotated out of the least significant bit of the operand are counted and the result stored in RA.





Format:

ROd RA, #n_to_shift

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 10, rotate		I/R = 0	n_to_shift				

ROd RA, Rp
ROd.WT RA, Rp

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 10,rotate		I/R = 1	L/R	W	0	Rp	

Encoding:

	0	1
I/R	immediate	register
L/R	left	right
W	shift	weight

Length/Cycles:
2 bytes, 4 + 1 per shift cycles

Condition Codes – weight not selected:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	*	Set according to last bit rotated out operand, cleared for a shift of zero.

Condition Codes – weight selected:

I	-	Not affected
N	0	Always cleared
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	0	Always cleared
C	*	Set according to least significant bit of result, i.e. parity of the n bits of the operand.

Example:
```
R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01BE] SP[8321] PS[02(....N...)]
01BD: D8 80       :ROL     R0,#0

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01C0] SP[8321] PS[00(........)]
01BF: D8 87       :ROL     R0,#7

R0[1A09] R1[9ABC] R2[9AFD] R3[FFFE] PC[01C2] SP[8321] PS[20(C.......)]
01C1: D8 A3       :ROL     R0,R3

R0[4682] R1[9ABC] R2[9AFD] R3[FFFE] PC[01C4] SP[8321] PS[00(........)]
01C3: D8 B3       :ROR     R0,R3

R0[1A09] R1[9ABC] R2[9AFD] R3[FFFE] PC[01C6] SP[8321] PS[20(C.......)]
01C5: D9 A2       :ROL     R1, R2

R0[1A09] R1[9357] R2[9AFD] R3[FFFE] PC[01C8] SP[8321] PS[22(C...N...)]
01C7: D9 AA       :ROL.WT  R1, R2

R0[1A09] R1[0003] R2[9AFD] R3[FFFE] PC[01CA] SP[8321] PS[20(C.......)]
01C9: D9 AA       :ROL.WT  R1, R2

R0[1A09] R1[0002] R2[9AFD] R3[FFFE] PC[01CC] SP[8321] PS[00(........)]
```

eXtended Rotate: ROXL, ROXR

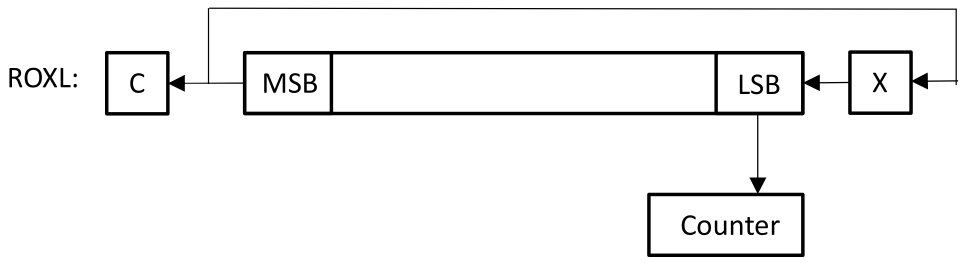
Operation:

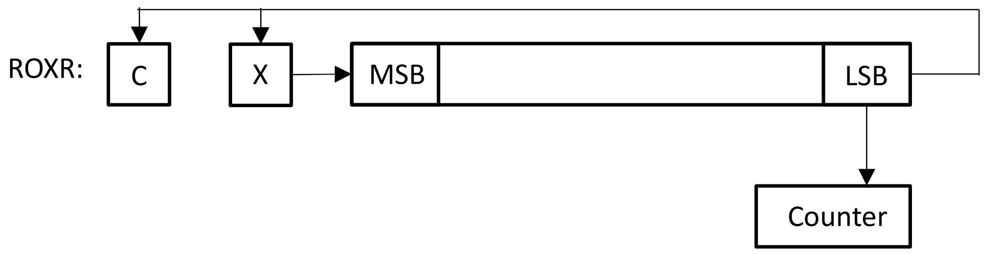
RA rotated by amount	⇒	RA
weight of n bits of RA	⇒	RA

Description:
The specified general register is rotated in the given direction for the required number of bits. The X bit is included in the rotation. The last bit rotated out of the operand (MSB if rotating left, LSB if rotating right) copied to C and X.
The shift amount may be specified in two ways:

1. as an immediate operand, this is a 5 bit signed value. (In the assembler use of the ROXR instruction is translated to ROXL and the immediate value is negated to compensate).
2. as a register, Rp, in which case the least significant 5 bits are used and treated as a signed value. These 5 bits are negated if the ROXR instruction was used.

If the shift count is specified by a register then the instruction may be qualified with .WT. In this case the number of 1 bits that are rotated out of the least significant bit of the operand are counted and the result stored in RA.





Format:

ROXd RA, #n_to_shift

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 11, X rotate		I/R = 0	n_to_shift				

ROXd RA, Rp
ROXd.WT RA, Rp

d may be L or R

7	6	5	4	3	2	1	0
Code = 0xD, immediate				1	0	RA	
fn = 11, X rotate		I/R = 1	L/R	W	0	Rp	

Encoding:

	0	1
I/R	immediate	register
L/R	left	right
W	shift	weight

Length/Cycles:
2 bytes, 4 + 1 per shift cycles

Condition Codes – weight not selected:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	*	Set according to last bit rotated out operand, not affected for a shift by zero
C	*	Set according to last bit rotated out operand, cleared for a shift by zero.

Condition Codes – weight selected:

I	-	Not affected
N	0	Always cleared
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	0	Always cleared
C	*	Set according to least significant bit of result, i.e. parity of the n bits of the operand.

Example:
```
R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01D8] SP[8321] PS[02(....N...)]
01D7: D8 C0       :ROXL           R0,#0

R0[1234] R1[9ABC] R2[9AFD] R3[FFFE] PC[01DA] SP[8321] PS[00(........)]
01D9: D8 C7       :ROXL           R0,#7

R0[1A04] R1[9ABC] R2[9AFD] R3[FFFE] PC[01DC] SP[8321] PS[30(CX......)]
01DB: D8 E3       :ROXL           R0,R3

R0[4681] R1[9ABC] R2[9AFD] R3[FFFE] PC[01DE] SP[8321] PS[00(........)]
01DD: D8 F3       :ROXR           R0,R3

R0[1A04] R1[9ABC] R2[9AFD] R3[FFFE] PC[01E0] SP[8321] PS[30(CX......)]
01DF: D9 E2       :ROXL           R1, R2

R0[1A04] R1[3357] R2[9AFD] R3[FFFE] PC[01E2] SP[8321] PS[30(CX......)]
01E1: D9 EA       :ROXL.WT        R1, R2

R0[1A04] R1[0003] R2[9AFD] R3[FFFE] PC[01E4] SP[8321] PS[20(C.......)]
01E3: D9 EA       :ROXL.WT        R1, R2

R0[1A04] R1[0002] R2[9AFD] R3[FFFE] PC[01E6] SP[8321] PS[00(........)]
```

SQRT

Operation:

$\lfloor \sqrt{R1} \rfloor$	$\Rightarrow$ R0
R1 – R0	$\Rightarrow$ R1
0	$\Rightarrow$ R3

Description:
The square root of R1 (rounded down) is calculated and stored in R0. The error is stored in R1. R3 is set to zero. The arithmetic condition flags are cleared.

Format:

7	6	5	4	3	2	1	0
0xF, miscellaneous				0	1	1	1

Length/Cycles:
1 byte, 18 cycles

Condition Codes:

I	-	Not affected
N	0	Set to zero
Z	0	Set to zero
V	0	Set to zero
X	0	Set to zero
C	0	Set to zero

Example:
```
R0[0002] R1[1705] R2[0006] R3[5678] PC[014A] SP[8321] PS[10(.X......)]
0149: F7       :SQRT       R1

R0[004C] R1[0075] R2[0006] R3[0000] PC[014B] SP[8321] PS[00(........)]
```

St

For store instructions see Ld, St.

SUB RA, RB

Operation:

RA - RB $\Rightarrow$ RA

Description:
The contents of the source general purpose register are subtracted from the destination general purpose register. The condition codes are set according to the result. The source and destination registers must be different.

Format:

7	6	5	4	3	2	1	0
Code = 0x6, SUB				RB		RA	

RA $\neq$ RB

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

Example:
```
R0[8000] R1[D4CC] R2[0001] R3[A6AB] PC[007D] SP[233B] PS[7B(CXV.NID.)]
007C: 65     :SUB   R2,R1

R0[8000] R1[D4CC] R2[2B35] R3[A6AB] PC[007E] SP[233B] PS[71(CX...ID.)]
```

SUBX R0, R1

Operation:

R0 - R1 - X ⇒ R0

Description:
The contents R1 and the X flag are subtracted from R0 and the result is stored in R0. The condition codes are set according to the result.

Format:

7	6	5	4	3	2	1	0
Code = 0xF, miscellaneous				1	1	0	1

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Cleared if result is non-zero, otherwise unchanged
V	*	Set if overflow is generated. Cleared otherwise
X	*	As per carry bit.
C	*	Set if carry is generated. Cleared otherwise

NB. If the Z flag is set before the start of an operation then it will achieve correct test for a zero result after completing an extended precision operation.

Example:
```
R0[151A] R1[002B] R2[552B] R3[5678] PC[00F8] SP[8321] PS[10(.X......)]
00FE: FD     :SUBX  R0,R1

R0[14EE] R1[002B] R2[552B] R3[5678] PC[00F9] SP[8321] PS[00(........)]
00FF: FD     :SUBX  R0,R1

R0[14C3] R1[002B] R2[552B] R3[5678] PC[00FA] SP[8321] PS[00(........)]
```

SXT RA

Operation:

sxt(RA) ⇒ RA

Description:
The sign bit (bit 7) of the low byte of a general purpose register is replicated through the upper byte. The condition codes are set according to the new value of the register.

Format:

7	6	5	4	3	2	1	0
Code = 0x0, MOVE				RA		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[F059] R1[00A2] R2[41FB] R3[A3C0] PC[67CA] SP[233B] PS[C0(......DU)]
67C9: 05     :SXT    R1

R0[F059] R1[FFA2] R2[41Fb] R3[A3C0] PC[67CB] SP[233B] PS[C2(....N.DU)]
```

Example:
```
R0[F059] R1[0074] R2[41FB] R3[A3C0] PC[67CA] SP[233B] PS[42(....N.D.)]
67C9: 05     :SXT    R1

R0[F059] R1[0074] R2[41FB] R3[A3C0] PC[67CB] SP[233B] PS[40(......D.)]
```

TEST RA

Operation:

PS set according to RA

Description:
The flags are set according to the contents of the source general purpose

Format:

7	6	5	4	3	2	1	0
Code = 0x1, AND				RA		RA	

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[0000] R1[0001] R2[FFFF] R3[7B75] PC[006B] SP[233B] PS[43(....NID.)]
006A: 10     :TEST  R0

R0[0000] R1[0001] R2[FFFF] R3[7B75] PC[006C] SP[233B] PS[45(...Z.ID.)]
006B: 15     :TEST  R1

R0[0000] R1[0001] R2[FFFF] R3[7B75] PC[006D] SP[233B] PS[41(.....ID.)]
006C: 1A     :TEST  R2

R0[0000] R1[0001] R2[FFFF] R3[7B75] PC[006E] SP[233B] PS[43(....NID.)]
```

TRAP

Operation:

SP – 1	⇒ SP
PS	⇒ [SP]
0	⇒ I
SP – 2	⇒ SP
PC + 1	⇒ [SP]
address	⇒ PC

Description:
The stack pointer is decremented by two. The address of the next instruction is stored in memory at the location pointed to by the stack pointer. The stack pointer is decremented by one. The PS is stored in memory at the location pointed to by the stack pointer. The interrupt flag is cleared. The program counter is set to 0x000C.

Format:

7	6	5	4	3	2	1	0
Code = 0xC, pushpop				1	1	0	1

Length/Cycles:
1 byte; 6 cycles

Condition Codes:

I	0	Always cleared
N	-	Not affected
Z	-	Not affected
V	-	Not affected
X	-	Not affected
C	-	Not affected

Example:
```
0060: 00 78 00 78 56 FF 34 2B 55 FF 55 DE 55 55 DE FF

R0[015E] R1[002B] R2[552B] R3[5678] PC[00E0] SP[0068] PS[01(.....I..)]
00DF: CD    :TRAP #3

R0[015E] R1[002B] R2[552B] R3[5678] PC[000D] SP[0066] PS[00(........)]

0060: 00 78 00 78 56 01 E0 00 55 FF 55 DE 55 55 DE FF
```

XOR RA, RB

Operation:

RB ^ RA ⇒ RA

Description:
The exclusive or of the source and destination general purpose registers is written into the destination. The condition codes are set according to the result. The source and destination registers may be the same.

Format:

7	6	5	4	3	2	1	0
Code = 0x2, XOR				RB		RA	

CLR RA is an alias for XOR RA, RA

Length/Cycles:
1 byte, 1 cycle

Condition Codes:

I	-	Not affected
N	*	Set if MSB set, cleared otherwise
Z	*	Set if result is zero, cleared otherwise
V	0	Always cleared
X	-	Not affected
C	0	Always cleared

Example:
```
R0[0000] R1[0001] R2[FFFF] R3[7B75] PC[006E] SP[233B] PS[43(....NID.)]
006D: 2D      :XOR   R1,R3

R0[0000] R1[7B74] R2[FFFF] R3[7B75] PC[006F] SP[233B] PS[41(.....ID.)]
006E: 2F      :CLR   R3

R0[0000] R1[7B74] R2[0000] R3[0000] PC[0070] SP[233B] PS[45(...Z.ID.)]
```